Пояснювальна записка
до дипломного проекту
на тему: «Моделювання дискретного автомату системи керування засобами CoDeSys»

Київ – 2019 рік

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

АА – Абстрактний автомат;

КА – Кінцевий автомат;

МЕК – Міжнародна електротехнічна комісія;

IL – Instruction List – список інструкцій;

ST – Structured Text – структурований текст;

SFC – Sequential Function Chart – послідовні функційні діаграми;

FBD – Function Block Diagram – функціонально-блокові діаграми;

LD – Ladder Diagram – сходинкова діаграма, мова релейно-контактнихсхем.

ВСТУП

Незважаючи на стрімкий розвиток технологій, надійні методи прикладного програмування і мистецтво програмування розвиваються вкрай повільно. Шлях, що дозволяє подолати труднощі, властиві розробці програмного забезпечення, пролягає через використання відповідних методологій. В епоху пакетної обробки безроздільно царювали алгоритми. Цей період можна назвати ерою «блок-схем». При побудові серверних додатків, що відповідають на запити, велику роль відіграє «відсутність стану» – немає потреби зберігати стан між двома послідовними запитами. При побудові успішного інтерактивного додатка, керованого подіями, багато що залежить від того, чи продумана модель управління станами.

У наш час широкого розповсюдження набула теорія автоматів. Вона відноситься до числа ключових розділів сучасної кібернетики. Теорія автоматів безпосередньо пов'язана з математичною логікою, теорією алгоритмів, теорією формальних граматик. Універсальність і порівняльна простота автоматів зумовили широке використання автоматних моделей на практиці.

Дискретний автомат – досить зручна концепція, яку доцільно використовувати для структурування додатків. Продумане застосування кінцевих автоматів полегшує організацію та супровід, як логіки призначеного для користувача інтерфейсу, так і логіки програми.

Дискретний автомат, як зазначалося вище, є засобом формальної структуризації програми. Замість того, щоб використовувати всі змінні програми як розширеного визначення його стану, кінцевий автомат створює єдину змінну, в якій зберігається інформація про стан програми. Такою змінною зазвичай є елемент деякої множини дійсних станів, що визначається у глобальному контексті або рівні класу. Актуальність підходу, що використовує кінцеві автомати, обумовлена тим, що він дозволяє в явному вигляді визначити дійсні стани для деякого аспекту вашого застосування і задати відповідні варіанти

поведінки при переходах додатка з одного стану в інший. Кінцеві автомати використовуються для формування набору взаємопов'язаних змінних або варіантів поведінки і їх логічної організації, що полегшує обробку станів.

Актуальність теми даної роботи випливає з того, що застосування кінцевих автоматів полегшує організацію та супровід логіки розроблених програм та оптимізує процес розробки програмного забезпечення.

Предметом дослідження є дискретні автомати в системах керування.

Об'єктом дослідження є модель дискретного автомату на прикладі системи керування генераторною установкою.

Метою дипломного проекту є моделювання дискретного автомату для системи керування генераторною установкою в програмному комплексі CoDeSys на основі існуючих моделей дискретних автоматів.

Для досягнення поставленої мети визначені такі завдання дослідження:

а) Вивчити предметну область, а саме – теорію дискретних автоматів та оглянути існуючі моделі дискретних автоматів;

б) Обрати спосіб задання дискретного автомату, який найбільше підходить для досягнення мети роботи;

в) Розглянути особливості роботи в середовищі CoDeSys та ознайомитися з мовами, що використовуються у процесі моделювання;

г) Побудувати діаграму станів та SFC-діаграму для системи керування генераторною установкою;

д) Виконати імітаційне моделювання дискретного автомату в середовищі CoDeSys.

1 ОГЛЯД МОДЕЛЕЙ ДИСКРЕТНИХ АВТОМАТІВ В СИСТЕМАХ КЕРУВАННЯ

1.1 Огляд теорії дискретного автомату

Автоматом називають дискретний перетворювач інформації, котрий приймає вхідні сигнали в дискретні моменти часу і з урахуванням свого колишнього стану формує вихідні сигнали і змінює свій стан [1].

Предметом теорії автоматів є вивчення математичних моделей перетворювачів дискретної інформації. У даній теорії вирішуються такі основні задачі: аналіз і синтез автоматів, визначення повноти, мінімізація і еквівалентні перетворення автоматів. Дамо коротку формулювання кожної з перерахованих задач [2, 3].

Задача аналізу. По заданому автомату описати його поведінку. Варіант постановки: по неповному опису автомата встановити деякі його властивості.

Задача синтезу. Побудувати автомат з наперед заданою поведінкою (алгоритмом функціонування). Завдання синтезу прийнято розглядати двояко: абстрактний синтез як побудову математичної моделі автомата і структурний синтез як розробку функціональної логічної схеми автомата.

Задача повноти. Нехай M – деяка множина автоматів. Визначити, чи володіє сукупність автоматів, що складають підмножину $M' \in M$, властивістю повноти. Іншими словами, чи співпадуть M' і M, якщо до всіх автоматів підмножини M' скінченне число разів застосувати операцію суперпозиції?

Задача мінімізації. Побудувати автомат, мінімальний заданому. Мінімальний автомат володіє найменшим числом компонентів моделі (зокрема, мінімальною потужністю множини так званих станів) і при цьому функціонально еквівалентний заданому автомату.

Задача еквівалентних перетворень. Визначити повну систему правил, що дозволяє перетворювати довільний автомат в будь-який еквівалентний йому автомат. Окремим випадком даної задачі є перехід від однієї моделі автомата до

іншого. Два автомата функціонально еквівалентні, якщо їх поведінка однакова під впливом одних і тих же послідовностей вхідних сигналів. У такому випадку говорять про збіг моделей поведінки двох автоматів.

У число основних понять теорії автоматів входять:

а) абстрактний автомат (АА);

б) композиція автоматів.

Поняття АА дозволяє розглядати дискретні пристрої з точки зору алгоритмів їх функціонування, тобто послідовностями дій, що реалізуються, по перетворенню дискретної інформації.

Абстрактним автоматом [4] називають модель, що складається з шести елементів (1.1):

$$N = \{S, X, Y, f, \varphi, S_0\}, \quad (1.1)$$

де $S = (s_1, s_2, \ldots, s_l)$ – алфавіт станів;

$X = (x_1, x_2, \ldots, x_n)$ – вхідний алфавіт автомата;

$Y = (y_1, y_2, \ldots, y_m)$ – вихідний алфавіт автомата;

f – функція переходів між станами;

φ – функція виходів;

S_0 – початковий стан;

Якщо множини S, X, Y – скінченні, то такий АА називають кінцевим автоматом (КА). Якщо хоча б одна з перерахованих множин не є скінченною, то такий АА називають нескінченним.

З метою класифікації автоматів розглядають ряд ознак, таких як визначеність функції переходів і функції виходів, однозначність заданих функцій, стійкість станів [5]. Перерахуємо види абстрактних автоматів, розподіливши їх за класифікаційними ознаками (рисунок 1.1).

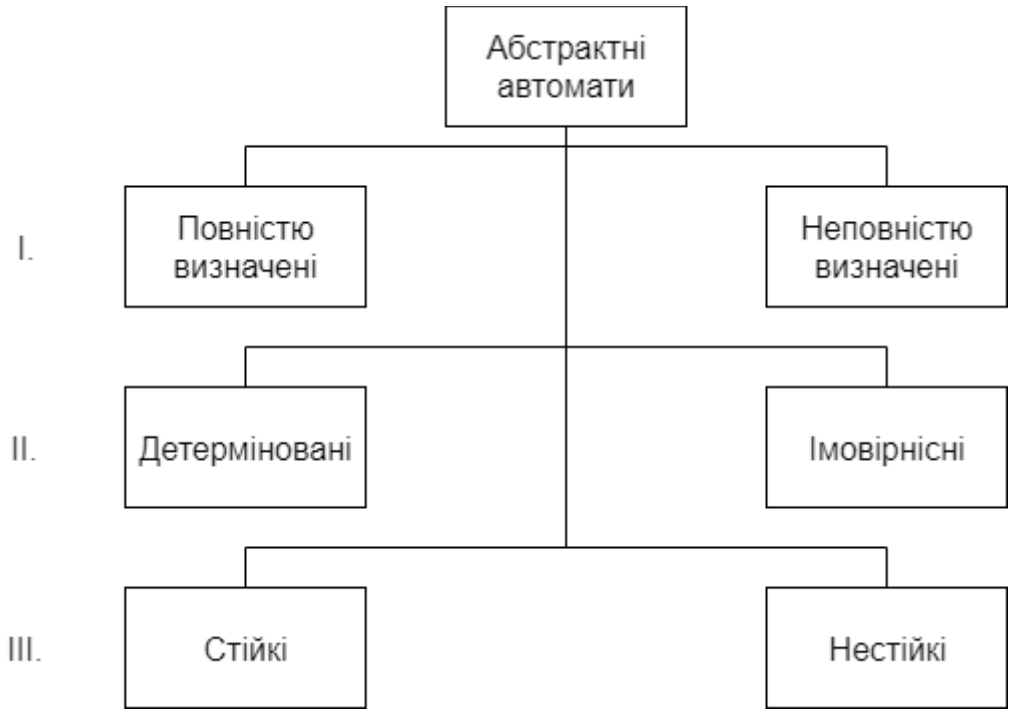

Рисунок 1.1 – Класифікація абстрактних автоматів

Розглянемо представлення абстрактного автомата, що зображене на рисунку 1.2.

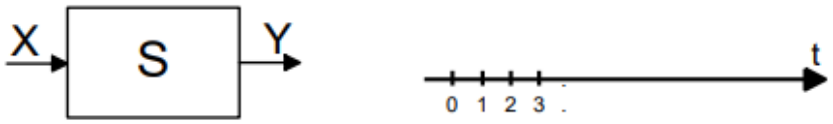

Рисунок 1.2 – Представлення абстрактного автомата

На наведенному вище рисунку t – дискретний час: t = nT, де T – інтервал (такт), що розділяє дискретні моменти часу; якщо T = 1, то t = n, тобто дискретний час зіставляється з упорядкованим рядом натуральних чисел.

Функціонування або поведінку автомата при заданих множинах і початковому внутрішньому стані повністю детерміноване і визначається функціями переходів і виходів [6, 7].

Функція переходів встановлює залежність внутрішнього стану автомата в наступний момент часу від стану входу і внутрішнього стану в даний момент

часу. Функція виходів встановлює залежність стану виходу автомата від стану входу і внутрішнього стану автомата.

Абстрактний автомат можна розглядати як «чорний ящик» з одним входом і одним виходом, з яким можна експериментувати, не знаючи, що знаходиться всередині [8].

Вихідний символ ($y_1 \in Y$) залежить не тільки від вхідного символу ($x \in X$), але і від того, в якому стані ($s_1 \in S$) знаходиться автомат. Автомат функціонує в дискретному часі; це означає, що елементи опису автомата задані тільки в згадані вище дискретні моменти.

Уявімо, що з деякого початкового, наприклад, нульового моменту часу на вхід автомата подаються вхідні символи, що утворюють вхідне слово деякої довжини (довжина будь-якого і-го слова вимірюється числом символів). На виході отримуємо вихідне слово тієї ж довжини (рисунок 1.3).

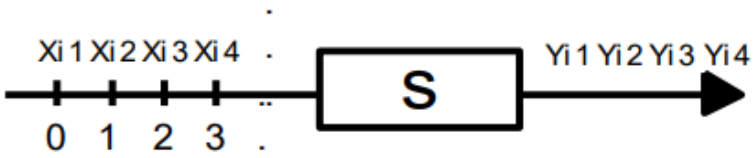

Рисунок 1.3 – Перетворення вхідного слова у вихідне

Отже, автомат може розглядатися як перетворювач вхідних слів у вихідні зі збереженням довжини слів. Символи алфавітів, присутні на вході і виході автомата, будемо також називати вхідними та вихідними сигналами [9].

1.2 Огляд моделей дискретних автоматів

1.2.1 Автомати Мілі і Мура та зв’язок між ними

На практиці широкого поширення набули дві основні моделі, що описують

функціонування АА. Їх виділяють за способом формування функцій виходів. Одна із цих моделей - модель Мілі. Закони функціонування автомата Мілі представлені системою рівнянь (1.2):

$$\begin{cases} q(t+1) = \delta\big(q(t), x(t)\big), \\ y(t) = \lambda\big(q(t), x(t)\big), \end{cases} \tag{1.2}$$

де t – поточний момент часу;

t+1 – наступний момент часу;

q(t+1) – стан автомата в наступний момент часу;

q(t), x(t), y(t) – елементи опису автомата в поточний момент часу.

З наведених вище рівнянь видно, що аргументами характеристичних функцій є поточне значення вхідного сигналу і поточний стан. Функціональна схема не відрізняється від схеми абстрактного автомата.

Друга, широко поширена, модель – модель Мура. Закони функціонування автомата Мура представлені системою рівнянь (1.3):

$$\begin{cases} q(t+1) = \delta\big(q(t), x(t)\big), \\ y(t) = \lambda\big(q(t)\big). \end{cases} \tag{1.3}$$

Як бачимо, в автоматі Мура функція виходів визначає значення вихідного символу тільки по одному аргументу – стану автомата. Цю функцію називають також функцією міток, так як вона кожному стану автомата ставить мітку на виході.

У моделі Мура вихідний сигнал явно залежить тільки від стану, а побічно – і від вхідного сигналу. В автоматах Мура вихідні впливи записані на станах, а в автоматі Мілі – на переходах.

Зауважимо, що автомат Мілі по відношенню до автомату Мура відстає на один дискретний момент часу по вхідному сигналу.

Будь-який автомат можна спроектувати з тієї чи іншої моделі. Автомат Мура переходить в автомат Мілі, якщо для всіх переходів у стан поставити вихідні впливи цього стану. Після таких перетворень отримаємо еквівалентний автомат Мілі.

Однак, щоб перетворити автомат Мілі в автомат Мура такий алгоритм не підходить, тому що в один стан можуть вести різні переходи. Для цього можна просто додати нові стани, встановлюючи необхідні відповідності.

Розглянемо послідовність дій у разі перетворення автомата Мура на еквівалентний йому автомат Мілі.

Припустимо, що початковий автомат Мура заданий множиною вхідних, вихідних сигналів, множиною внутрішніх станів та функціями переходу та виходу (1.4).

$$G_A = \{X_A, Y_A, S_A, f_A, \varphi_A\}. \quad (1.4)$$

Створюємо автомат Мілі, заданий аналогічно (1.5):

$$G_Б = \{X_Б, Y_Б, S_Б, f_Б, \varphi_Б\}. \quad (1.5)$$

При цьому необхідно забезпечити виконання таких умов (1.6):

$$X_A = X_Б;\ Y_A = Y_Б;\ S_A = S_Б;\ f_A = f_Б;\ \varphi_A = \varphi_Б. \quad (1.6)$$

Еквівалентність функцій виходів автоматів визначається так: якщо для автомата Мура маємо функцію переходів $f_A(x_m, s_n) = S_j$ і його функція виходів $\varphi_A(S_j) = y_l$, то і в автоматі Мілі функція виходів $\varphi_Б(x_m, Sj)$ також повинна формувати вихідний сигнал y_l.

Перехід від автомата Мура за табличного способу задання G_A до автомата Мілі $G_Б$, коли таблиці переходів збігаються, реалізується шляхом заміни символу

s_j на перетині стовпця x_m та рядка s_n в таблиці переходів символом вихідного сигналу y_l, який відмічає рядок s_n в таблиці пепреходів автомата Мура. Унаслідок послідовно виконаних операцій одержуємо суміщену таблицю переходів та виходів автомата Мілі. Із самого принципу побудови автомата Мілі $G_Б$ очевидна його еквівалентність початковому автомату Мура G_A.

Для вирішення зворотної проблеми – перетворення автомата Мілі на еквівалентний автомат Мура існує простий конструктивний метод: Реалізація кожного стану s_j автомата Мілі, перехід у який супроводжується формуванням різних вихідних сигналів, здійснюється за допомогою сукупності станів $s_j^1, s_j^2, \dots, s_j^k$, кожному з яких відповідає тільки один вихідний сигнал. У позначеній таблиці переходів автомата Мура фіксують нові введені стани. При цьому, очевидно, кількість внутрішніх станів автомата Мура буде більшою, ніж для еквівалентного йому автомата Мілі.

1.2.2 С-автомат

Під абстрактним С – автоматом розуміють математичну модель дискретного пристрою, яка визначається восьмикомпонентним вектором (1.7):

$$L = \{X, S, Y_1, Y_2, f, \varphi_1, \varphi_2, s_0\}, \quad (1.7)$$

де $X = (x_1, x_2, \dots, x_n)$ – вхідний алфавіт автомата;

$S = (s_1, s_2, \dots, s_m)$ – алфавіт внутрішніх станів;

$Y_1 = (y_{11}, y_{12}, \dots, y_{1k})$ – вихідний алфавіт I типу;

$Y_2 = (y_1, y_2, \dots, y_{2l})$ – вихідний алфавіт II типу;

f – функція переходів автомата;

φ_1 – функція виходів для сигналів I типу;

φ_2 – функція виходів для сигналів II типу;

s_0 – початковий стан;

Абстрактний автомат С – автомат можна зобразити у вигляді пристрою (рисунок 1.4), який має один вхід і два вихідні канали, які одночасно реалізують різні функції виходів: I і II роду.

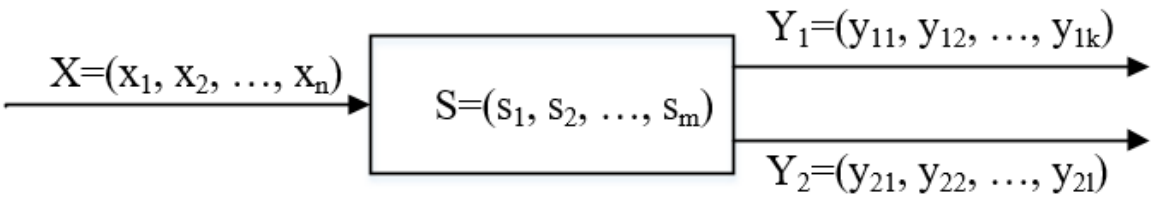

Рисунок 1.4 – Абстрактний С - автомат

Від С – автомата можна перейти до автоматів Мілі, Мура і навпаки. З іншого боку, часто виявляється зручним використовувати одночасно вихідні сигнали обох типів. Так, під час проектування автомату частина сигналів блоку керування, таких як передача з регістра до регістра, нарощення лічильників тощо, має тривалість, порівняну із вхідним сигналом, тактованим синхроімпульсами. У той же час, наприклад, сигнали, які надходять на комбінаційний суматор або дешифратор, значно триваліші. Ці сигнали зручніше ототожнювати з внутрішнім станом.

1.2.3 Мережі Петрі

Початок теорії мереж Петрі поклав німецький вчений Карл Петрі, який в 1962 році запропонував мережеву модель асинхронних інформаційних потоків в дискретних системах і застосував її для дослідження кінцевих автоматів. Досить швидко мережі Петрі завоювали широку популярність як вельми зручний формальний апарат для представлення і вивчення взаємодії паралельних подій і процесів в дискретних S_4 системах.

Мережа Петрі – це орієнтований граф особливого виду, що складається з двох функціонально різних типів вузлів (або вершин) – позицій і переходів.

Позиції і переходи з'єднуються між собою дугами таким чином, що кожна дуга пов'язує позицію і перехід, але не дві позиції або два переходи. Це означає, що мережа Петрі в математичному аспекті можна розглядати як двочастковий граф, в якому одна множина вузлів складається з позицій, а інша множина вузлів – з переходів. Однієї і тієї ж позиції і одному і тому ж переходу може бути інцидентне довільне (кінцеве) число дуг. Формально структуру мережі Петрі можна задати четвіркою елементів (1.8):

$$PN = (P, T, Pre, Post), \quad (1.8)$$

де $P = \{p_1, p_2, \ldots, p_n\}$ – скінченна множина позицій, $p > 0$;

$T = \{t_1, t_2, \ldots, t_n\}$ – скінченна множина переходів, $t > 0$;

Pre – вхідна функція;

Post – вихідна функція.

Множини P і T не перетинаються, тобто $P \cap T = 0$, і кожна з них непорожня. Функція Pre для кожної пари (позиція, перехід) визначає кратність, або вагу орієнтованої дуги, що веде з позиції в перехід. Структура мережі Петрі визначається її позиціями, переходами вхідної та вихідної функцій.

Формально Pre: P x T –> N є відображенням декартового добутку множин P і T у множину чисел N = {0, 1,2, ...}.

Функція Post визначає для кожної пари (перехід, позиція) кратність дуги, що веде з переходу в позицію.

Формально Post: P x T –> N, причому кратність, або вага дуги, що веде з переходу t_k в позицію p_p є $Post(t_k, p) = w(t_k, p)$. Зауважимо, що кратна дуга може представлятися відповідним числом дуг з кратністю 1.

Дуже часто мережу Петрі представляється графічно. При цьому позиції зображуються кружками, а переходи – відрізками прямих. Повні набори позицій і переходів з конкретною системою з'єднання певних позицій і переходів – це

граф мережі Петрі. Дуги в мережах Петрі спрямовані, причому кожна дуга пов'язує вершини тільки різних класів (рисунок 1.5).

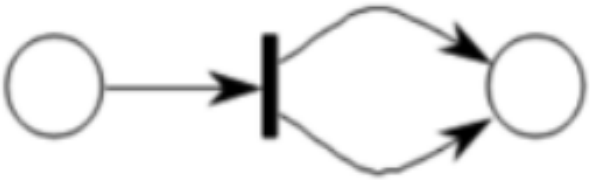

Рисунок 1.5 – Правильно побудована мережа Петрі

Назвемо вхідними позиціями деякого конкретного переходу ті позиції, з яких виходять дуги, що входять в даний перехід. Відповідно, вихідними позиціями назвемо позиції, в які входять дуги, які виходять із даного переходу.

На рисунках 1.6, 1.7 позиції позначені літерою p з індексом, а переходи – буквою t з індексом. Рисунок 1.6 є зображенням одинарної мережі Петрі, а рисунок 1.7 – мережі Петрі з кратними дугами, так як позиція p_x з'єднана з переходом t_1 двома дугами, а перехід t_2 з позицією p_5 – трьома дугами.

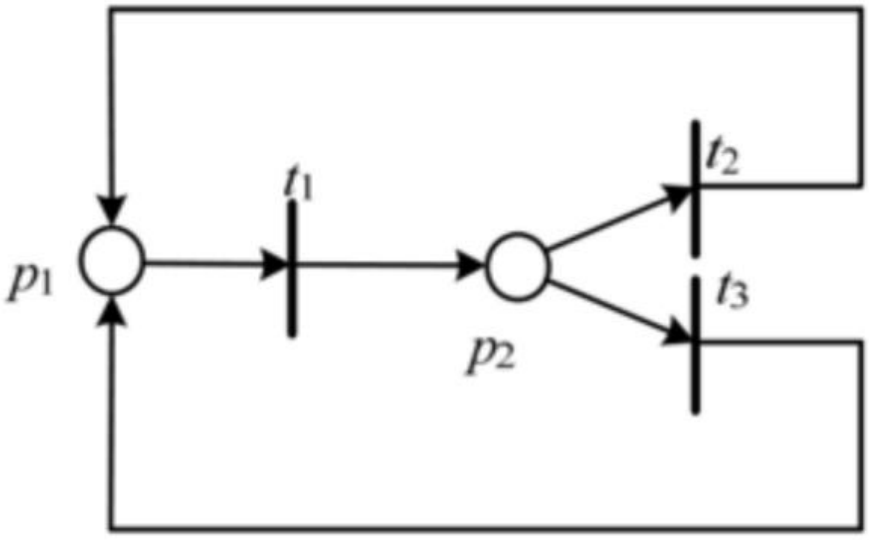

Рисунок 1.6 – Приклад одинарної мережі Петрі

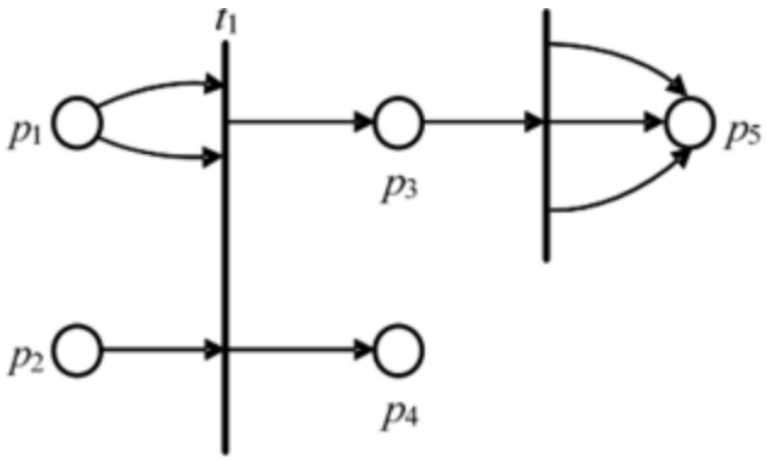

Рисунок 1.7 – Приклади мережі Петрі з кратними дугами

Умова в мережі Петрі має ємність: умова не виконана (ємність дорівнює 0), виконано (ємність дорівнює 1), умова виконана з х-кратним запасом (ємність дорівнює n, де n – ціле додатне число). Певні поєднання умов дозволяють реалізуватися деякому події (передумова події), а реалізація події змінює деякі умови (постумови події), тобто події взаємодіють з умовами, а умови – з подіями. Таким чином, для подання алгоритмів досить визначати дискретні системи як структури, утворені з елементів двох типів: подій і умов.

Маркування – це розміщення по позиціях мережі Петрі фішок, зображуваних на графі мережі Петрі точками. Фішки використовуються для визначення виконання мережі Петрі. Кількість фішок в позиції при виконанні мережі Петрі може змінюватися від 0 до нескінченності (таблиця 1.1).

Таблиця 1.1 – Символи розмітки виконання умов

Зміст умови	**Символ**
Умова р не виконана	
Умова р виконана	
Умова р має ємність 3	

Фішка може перебувати в тій чи іншій позиції мережі, причому одночасно в мережі може існувати багато фішок, розкиданих по різних позиціях. Різні маркування мережі Петрі характеризують стани відповідної їй динамічної системи, причому динаміка змін станів моделюється рухом фішок по позиціях. Неформально роботу мережі можна уявити як сукупність локальних дій, які називаються спрацьовуваннями переходів. Вони відповідають реалізаціям подій і призводять до зміни розмітки позицій, тобто до локальної зміни умов в системі. Якщо кожна з вхідних позицій переходу містить щонайменше одну мітку, то перехід t може спрацювати (збуджений).

При спрацьовуванні переходу з кожної його позиції видаляється одна мітка, а в кожну вихідну позицію додається одна мітка.

Мережа Петрі, що містить хоча б одну фішку в який-небудь позиції, називається маркованою, або розміченою, мережею Петрі. Таким чином, розмічена (маркована) мережа Петрі може бути формально задана як (1.9)

$$\mathrm{PN} = (\mathrm{P}, \mathrm{T}, \mathrm{Pre}, \mathrm{Post}, \mathrm{M}), \qquad (1.9)$$

де M – функція, яка для кожної позиції мережі Петрі визначає число фішок, які перебувають в позиції.

Функція, або вектор розмітки, повністю визначає поточний стан мережі. Маркування відображається відповідним числом точок в кожній позиції. Якщо фішок багато (більше трьох), то їх кількість відображається числом. Мережа Петрі виконується за допомогою запуску переходів. Перехід може бути запущений в тому випадку, коли він дозволений, тобто якщо кожна з його вхідних позицій містить число фішок, одна з, ніж число дуг з неї в даний перехід. Процедура запуску полягає у видаленні з кожної вхідної позиції переходу числа фішок, рівного числу дуг з неї, і в виставленні в кожної вихідний позиції числа фішок, рівного числу дуг, що входить до неї.

В результаті спрацьовування переходу з кожної його вхідний позиції

витягується число фішок, яка дорівнює кількості дуг, які з'єднують позицію з переходом, а в кожну вихідну позицію спрацював переходу додається число фішок, яка дорівнює кількості дуг, що ведуть з переходу в позицію. На рисунку 1.7 зображений приклад розміченої мережі.

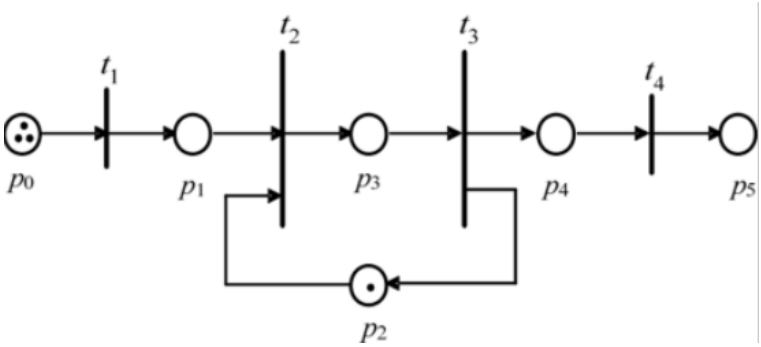

Рисунок 1.7 – Приклад розміченої мережі

Одна з основних проблем в теорії мереж Петрі – завдання про скінченність функціонування мережі (про досягнення тупикової розмітки), тобто у відповіді на питання, чи існує така послідовність спрацьовування переходів, яка призводить мережу до тупикової розмітки, до такої, при якій жоден перехід не може спрацювати (рисунок 1.8).

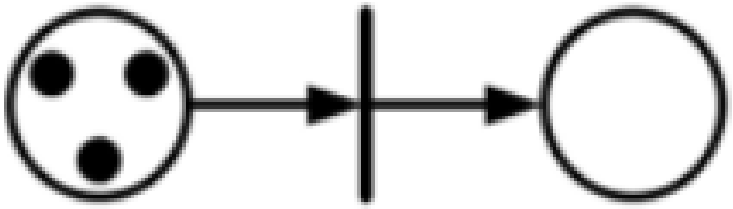

Рисунок 1.8 – Мережа, що приводить до тупикової розмітки

Основними властивостями мережі Петрі є:

а) обмеженість мережі Петрі – властивість мережі, число міток якої в будь-якій позиції мережі не може перевищити деякого значення K. Якщо в жодній позиції мережі при будь-якій послідовності спрацьовувань переходів

кількість фішок не перевищує деякого K, то таку мережу називають K-обмеженою (рисунок 1.9);

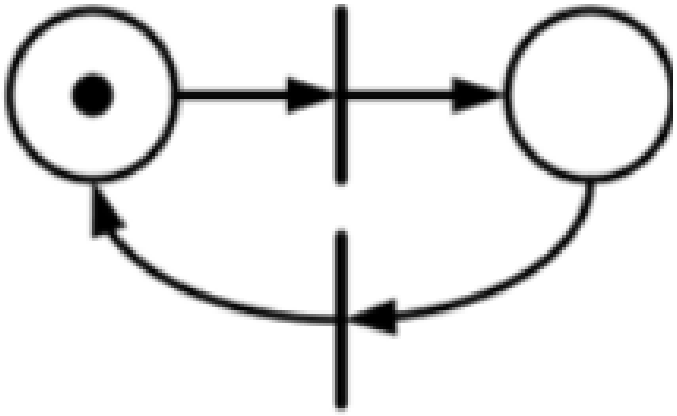

Рисунок 1.9 – Обмежена мережа

б) безпеку мережі Петрі – є окремий випадок обмеженості, K = 1;

в) збереженість мережі Петрі – є сталість завантаження ресурсів, коли $\Sigma A_i N_i$ постійна. Де N_i – число маркерів в і-тій позиції, A_i – ваговий коефіцієнт;

г) досяжність мережі Петрі – можливість переходу мережі з одного заданого стану (що характеризується розподілом міток) в інше;

д) жвавість мережі Петрі – можливість спрацьовування будь-якого переходу при функціонуванні об'єкта, що моделюється.

В основі дослідження перерахованих властивостей лежить аналіз досяжності. Методи аналізу властивостей мереж Петрі засновані на використанні графів досяжних (покривають) маркувань, вирішенні рівняння станів мережі і обчисленні лінійних інваріантів позицій і переходів. Застосовуються також допоміжні методи редукції, що дозволяють зменшити розмір мережі Петрі зі збереженням її властивостей, і декомпозиції, що розділяють вихідну мережу на підмережі. При моделюванні складних обчислювальних систем на декількох ієрархічних рівнях мережі Петрі можуть представлятися цілими підмережами. Окремими елементами в цьому випадку є прямокутники, що представляють непримітивні події, а планки представляють

примітивні події.

Нехай є розмічена мережа Петрі з вектором розмітки M і заданий вектор розмітки з тим же числом компонентів, що і вектор M. Завдання полягає у визначенні того, чи можливе досягнення розмітки (стану) M, якщо задана мережа стане працювати, починаючи з розмітки (стану) M. При цьому розмітка M' може представляти деякий бажаний або небезпечний стан системи, що моделюється мережею Петрі, так що можливість його досягнення зі стану M повинна бути забезпечена або, навпаки, виключена.

Для багатьох мереж Петрі завдання досяжності може бути вирішене побудовою орієнтованого дерева досяжності. Кожен вузол (вершина) в цьому дереві відповідає певному вектору розмітки розглянутої мережі Петрі. Коренем дерева досяжності служить вузол, відповідний початковій маркуванні M.

За допомогою мереж Петрі можна моделювати широкий клас систем, представляючи належним чином взаємодія різних процесів, які можуть виникнути в системі. Як зазначалося раніше, найбільш часто мережі Петрі застосовують при моделюванні систем, що включають паралельні дії.

Виділяють чотири типи завдань дослідження об'єктів за допомогою мереж Петрі:

а) інтерпретація (програмування об'єкта), пов'язана з адекватним поданням об'єкта, що моделюється відповідною мережею Петрі;

б) програмування моделі в конкретній операційному середовищі;

в) дослідження моделі;

г) крос-трансляція з мови мереж Петрі на мови програмування.

Існує багато видів мереж Петрі. Нижче наведені найбільш розповсюджені з них:

а) тимчасова мережа Петрі – така мережа, де переходи мають вагу, що визначає тривалість спрацьовування (затримку);

б) стохастична мережа Петрі – мережа, в якій затримки є випадковими величинами;

в) функціональна мережа Петрі – мережа, в якій затримки визначаються як функції деяких аргументів, наприклад, кількості міток в будь-яких позиціях, стану деяких переходів;

г) кольорова мережа Петрі – мережа, в якій мітки можуть бути різних типів, що позначені кольорами, тип мітки може бути використаний як аргумент в функціональних мережах;

д) інгібіторна мережа Петрі – мережа, в якій можливі інгібіторні, тобто ті, що пригнічують, дуги, які забороняють спрацьовування переходу, якщо у вхідній позиції, пов'язаної з переходом інгібіторною дугою, знаходиться мітка;

е) ієрархічна мережа Петрі – мережа, яка містить немиттєві переходи, в які вкладені інші, можливо, також ієрархічні, мережі. Спрацьовування такого переходу характеризує виконання повного життєвого циклу вкладеної мережі;

ж) WF-мережі Петрі – підклас мереж Петрі, званий також мережами потоків робіт. WF-мережі використовуються для перевірки графів потоків робіт на наявність таких структурних конфліктів, як «тупики» і «недоліки синхронізації».

Таким чином, мережі Петрі дозволяють перейти від понять абстрактної асинхронної системи до динамічної структурі з подій і умов. Моделюючі можливості мереж Петрі та їх ефективність пояснюються насамперед тим, що мережа Петрі – це інтеграція графа і дискретної динамічної системи. Вона може бути статичною або динамічною моделлю подається з її допомогою об'єкта. При цьому відсутність строго фіксованого аналітичного підходу при визначенні відношеня вхід-вихід мережі робить цю систему алгоритмічно невизначеною і для імітаційних моделей.

Застосування мереж Петрі не обмежується тільки моделюванням процесів і динамічних систем. Вони з успіхом використовуються і в теоретичному програмуванні при вирішенні задач функціональної специфікації, верифікації програмного забезпечення, організації обчислювальних процесів, управління.

1.3 Способи задання автоматів

Щоб задати дискретний автомат, необхідно описати всі його елементи, тобто вхідний, внутрішній і вихідний алфавіти, а також функції переходів і виходів. Причому серед безлічі станів необхідно виділити саме той стан, в якому автомат знаходився в момент часу t = 0. Існують кілька способів опису роботи дискретних автоматів, але найбільш часто використовуються табличний, графічний і матричний [10].

1.3.1 Таблиця переходів

Машина станів може бути представлена як таблиця. Наприклад, в одному із різновидів таких таблиць кожен рядок відповідає за поточний стан, а кожен стовпець – за вхідних сигнал. На перетині поточного стану та вхідного сигналу записується до якого стану перейде машина, якщо перехід можливий.

Для автомата Мілі табличний спосіб задання зводиться до побудови двох таблиць: таблиці переходів і таблиці виходів.

Для автомата Мура таблиця переходів і таблиця виходів об’єднуються і додається рядок вихідних сигналів, що відповідають станам автомата.

Опис скінченного автомата Мілі за допомогою таблиці переходів представлено у таблиці 1.2, а опис автомата Мура – у таблиці 1.3.

Таблиця 1.2 – Таблиця переходів автомата Мілі

		z_k		
x_i	z_0	z_1	…	z_K
Переходи				
x_1	$\varphi(z_0, x_1)$	$\varphi(z_1, x_1)$	…	$\varphi(z_K, x_1)$
x_2	$\varphi(z_0, x_2)$	$\varphi(z_1, x_2)$	…	$\varphi(z_K, x_2)$

Продовження таблиці 1.2

…	…	…	…	…
x_I	$\varphi(z_0, x_I)$	$\varphi(z_1, x_I)$	…	$\varphi(z_K, x_I)$
Виходи				
x_1	$\psi(z_0, x_1)$	$\psi(z_1, x_1)$	…	$\psi(z_K, x_1)$
x_2	$\psi(z_0, x_2)$	$\psi(z_1, x_2)$	…	$\psi(z_K, x_2)$
…	…	…	…	…
x_I	$\psi(z_0, x_I)$	$\psi(z_1, x_I)$	…	$\psi(z_K, x_I)$

Таблиця 1.3 – Таблиця переходів автомата Мура

		$\psi(z_k)$		
x_i	$\psi(z_0)$	$\psi(z_1)$	…	$\psi(z_K)$
	z_0	z_1	…	z_k
x_1	$\varphi(z_0, x_1)$	$\varphi(z_1, x_1)$	…	$\varphi(z_K, x_1)$
x_2	$\varphi(z_0, x_2)$	$\varphi(z_1, x_2)$	…	$\varphi(z_K, x_2)$
…	…	…	…	…
x_I	$\varphi(z_0, x_I)$	$\varphi(z_1, x_I)$	…	$\varphi(z_K, x_I)$

У разі реалізації табличного способу автомат Мілі задається таблицями переходів (таблиця 1.4) і виходів (таблиця 1.5). Рядки в таблицях відповідають внутрішнім станам, а стовпці – вхідним сигналам. На перетині стовпця X_i та рядка S_j в таблиці переходів фіксується внутрішній стан автомата у наступну мить часу протягом якого він перейде із попереднього стану під впливом вхідного сигналу X_i (1.10).

$$S(t+1) = f[S_j(t), X_i(t)]. \quad (1.10)$$

А в таблиці виходів відображається відповідний цьому переходу вихідний сигнал (1.11).

$$Y(t) = \varphi[S(t), X(t)]. \quad (1.11)$$

Таблиця 1.4 – Таблиця переходів автомата

S(t)	X(t)	
	X_1	X_2
1	2	3
2	3	2
3	2	1

Таблиця 1.5 – Таблиця виходів автомата

S(t)	X(t)	
	X_1	X_2
1	y_1	y_2
2	y_3	y_1
3	y_3	y_1

Часто, задаючи автомати Мілі, ми застосовуємо єдину сумісну таблицю переходів і виходів, в якій на перетині стовпця X_i і рядка S_j нотуємо відповідний внутрішній стан у момент часу (t+1) і вихідний сигнал Y(t). Прикладом такої сумісної таблиці є таблиця 1.6, яка створена на основі таблиць 1.3 і 1.5. Для частково визначених автоматів Мілі та Мура у відповідних клітинках таблиць переходів і виходів ставиться риска.

Таблиця 1.6 – Сумісна таблиця переходів та виходів

S(t)	X(t)	
	X_1	X_2
1	$2,y_1$	$3,y_2$
2	$3,y_3$	$2,y_1$
3	$2,y_3$	$1,y_1$

Оскільки в автоматі Мура вихідний сигнал формально є функцією лише внутрішнього стану, то він задається однією так званою позначеною таблицею переходів, у якій кожному її рядку додається крім стану S_j , ще і вихідний сигнал Y(t), відповідний цьому станові. Приклад табличного опису автомата Мура наведено у таблиці 1.7.

Таблиця 1.7 – Позначена таблиця переходів

Y(t)	S(t)	X(t)	
		X_1	X_2
y_1	1’	2’	3’
y_1	2’	5’	2’
y_2	3’	4’	1’
y_3	4’	3’	2’
y_3	5’	3’	4’

Для задання С – автомата також застосовується табличний метод. При цьому таблиця переходів (таблиця 1.8) аналогічна таблиці переходів автомата Мілі, а в таблиці виходів С – автомата (таблиця 1.9) кожний стан позначений відповідним йому вихідним сигналом автомата Мура.

Таблиця 1.8 – Таблиця переходів С – автомата

S(t)	X(t)	
	X_1	X_2
1	6	4
2	6	3
3	4	5
4	3	5
5	4	5
6	1	2

Таблиця 1.9 – Таблиця виходів С – автомата

Y(t)	S(t)	X(t)	
		X_1	X_2
y_{21}	1	y_{11}	y_{12}
y_{21}	2	y_{11}	y_{11}
y_{23}	3	y_{12}	y_{11}
y_{23}	4	y_{11}	y_{12}
y_{22}	5	y_{12}	y_{11}
y_{22}	6	y_{12}	y_{11}

1.3.2 Метод представлення у вигляді графа

При іншому способі завдання скінченного автомата використовується поняття напрямленого графа [11, 12]. Автомат представляється орієнтованим зв'язним графом (орграф), вершини якого відповідають станам автомата, а дуги – переходам зі стану в стан. Дві вершини графа автомата (початковий стан і стан переходу S_j і S_i) з’єднуються дугою, напрям якої вказується стрілкою.

Дузі графа приписують відповідні значення вхідного та вихідного сигналів, якщо вони визначені і ставлять риску, коли вони не визначені. Якщо перехід автомата із стану S_j у стан S_i відбувся під впливом декількох вхідних сигналів, то відповідній дузі (S_i, S_j) присвоюють усі значення вхідних та вихідних сигналів.

Коли за допомогою граф-схем описують автомат Мура, то вихідний сигнал (1.12) нотується або всередині вершини, або біля неї.

$$Y(t) = \varphi[S(t)]. \qquad (1.12)$$

На рисунку 1.10 зображений граф автомата Мілі, а на рисунку 1.11 – Мура, які відповідають таблицям 1.6, 1.7.

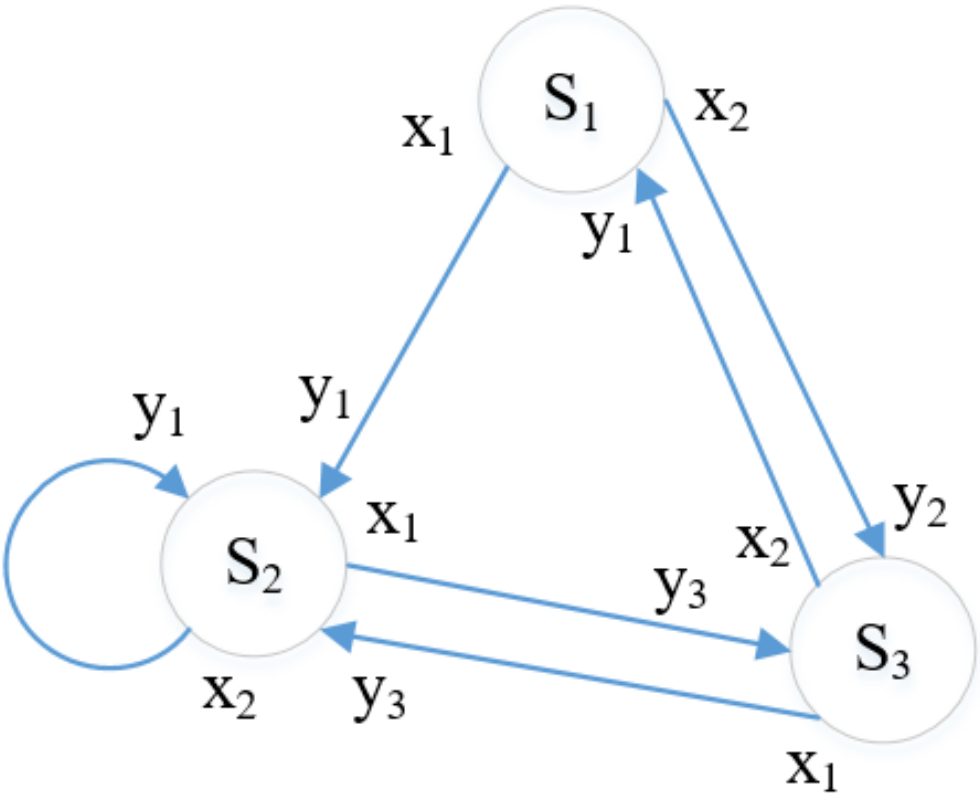

Рисунок 1.10 – Граф-схема автомата Мілі

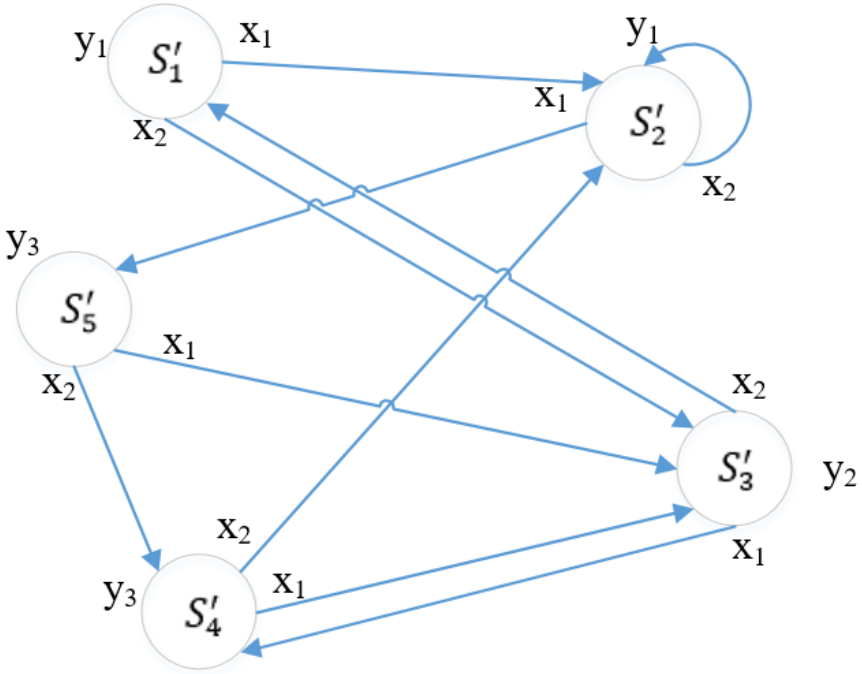

Рисунок 1.11 – Граф-схема автомата Мура

Серед послідовних автоматів часто зустрічаються автомати, на входи яких деякі комбінації вхідних сигналів не подаються. Такі комбінації називають забороненими, а відповідні автомати – частковими. Графи часткових автоматів містять вузли, в яких кількість дуг, що виходять, менша кількості букв вхідного алфавіту. Під час синтезу такого автомата необхідно довизначити функції переходів та виходів з метою максимального спрощення його структури. На рисунку 1.12 наведений приклад часткового визначення автомата Мілі.

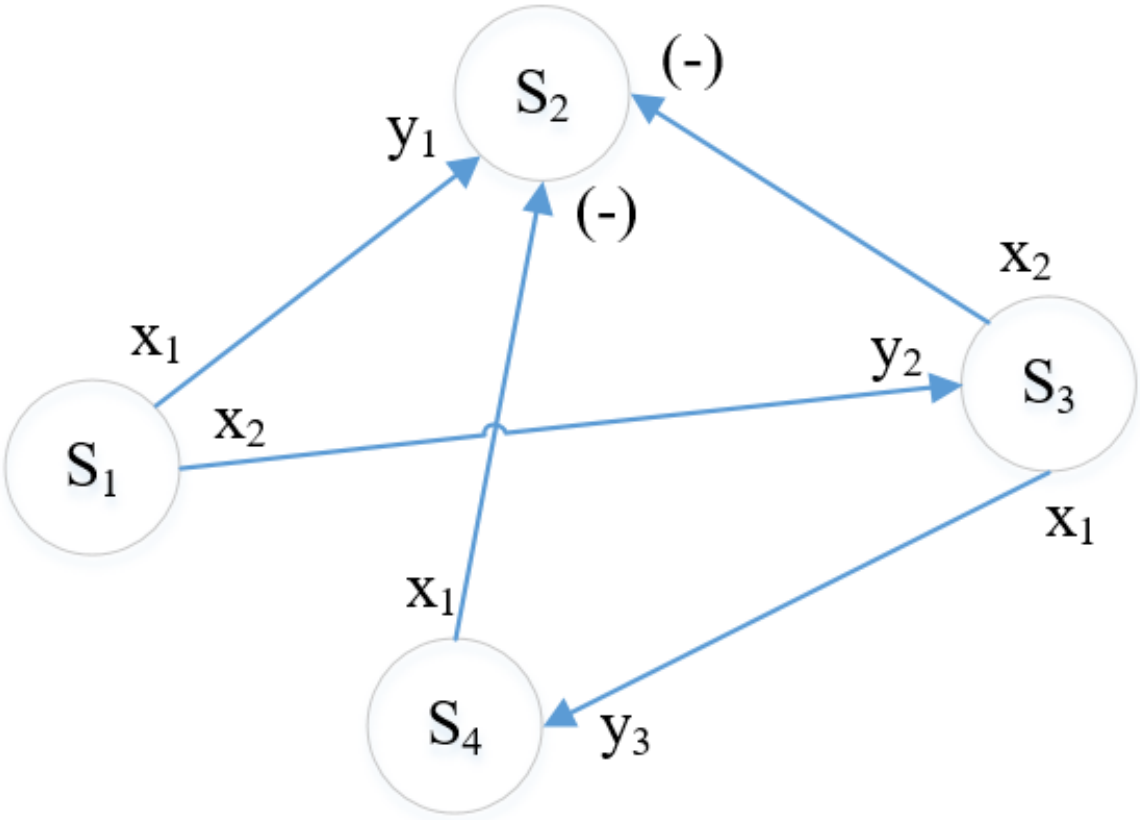

Рисунок 1.12 – Частково визначений автомат Мілі

Якщо ж необхідно за допомогою графа зобразити С – автомат, то в такому випадку сигналами I типу позначають дуги графа поряд із вхідними сигналами, а сигналами II типу – вершини, яким вони відповідають.

1.3.3 Матричний метод представлення автомата

Абстрактний автомат може задаватися також за допомогою так званої матриці з'єднань автомата – квадратної матриці $C = |c_{mn}|$, рядки якої відповідають початковим внутрішнім станам, а стовпці – станам у момент часу (t+1) [13].

Елемент $c_{mn} = x_i / y_j$, який знаходиться на перетині m-го рядка і n-го стовпця, у випадку автомата Мілі відповідає вхідному сигналу x_i, який забезпечує перехід автомата із внутрішнього стану S_m у стан S_n і вихідному сигналу y_j, який при цьому формується. Так, для автомата Мілі, заданого таблицею , матриця з'єднань (1.13) буде мати вигляд:

$$C = \begin{Vmatrix} - & x_1/y_1 & x_2/y_2 \\ - & x_2/y_1 & x_1/y_3 \\ x_2/y_1 & x_1/y_3 & - \end{Vmatrix} \qquad (1.13)$$

Якщо перехід із стану S_m у стан S_n відбувається внаслідок дії декількох сигналів, то елемент c_{mn} являє собою множину пар вхід/вихід для цього переходу, які з'єднуються законом диз'юнкції [14, 15].

За матричного задання автомата Мура елемент c_{mn} дорівнює множині вхідних сигналів на переході (S_m, S_n), а вихід також подається у вигляді відповідної матриці. Так, для автомата Мура, заданого таблицею, маємо (1.14).

$$C = \begin{Vmatrix} - & x_1 & x_2 & - & - \\ - & x_2 & - & - & x_1 \\ x_2 & - & - & x_1 & - \\ - & x_2 & x_1 & - & - \\ - & - & x_1 & x_2 & - \end{Vmatrix}; \qquad Y = \begin{Vmatrix} y_1 \\ y_1 \\ y_2 \\ y_3 \\ y_3 \end{Vmatrix}. \qquad (1.14)$$

ВИСНОВКИ ДО РОЗДІЛУ 1

У результаті виконання першого розділу дипломного проекту було описано та проаналізовано завдання теорії автоматів та поняття абстрактних автоматів. Було розглянуто основні моделі автоматів та методи їх задання. На основі дослідженого матеріалу можна зробити наступні висновки:

а) автомат – це дискретний перетворювач інформації, котрий приймає вхідні сигнали в дискретні моменти часу і з урахуванням свого колишнього стану формує вихідні сигнали і змінює свій стан;

б) теорія автоматів вивчає математичні моделі перетворювачів дискретної інформації – автоматів. Вона вирішує такі основні задачі, як аналіз і синтез автоматів, визначення повноти, мінімізація та еквівалентні перетворення автоматів;

в) абстрактний автомат – це модель, що складається з множини вхідних символів, множини станів автомата, множини вихідних символів, функцій переходів та виходів та початкового стану автомата;

г) автомати Мілі та Мура – дві основні моделі, що описують функціонування абстрактних автоматів. Вони відрізняються за способом формування функцій виходів. Окрім них є ще С – автомати та мережі Петрі;

д) для задання дискретного автомату необхідно описати всі його елементи. Найбільш розповсюджені способи опису їх роботи – це табличний, графічний і матричний.

2 CODESYS – ЯК ЗАСІБ МОДЕЛЮВАННЯ ДИСКРЕТНОГО АВТОМАТУ

2.1 Огляд програмного комплексу CoDeSys

Для моделювання дискретного автомату системи керування мною було обрано середовище CoDeSys. На сьогоднішній день CoDeSys (Controller Development System) – це найпопулярніший в світі апаратно незалежний комплекс для прикладного програмування ПЛК та вбудованих контролерів. Основним його компонентом є середовище програмування на мовах стандарту МЕК 61131-3. Комплекс працює на комп'ютері. Програми компілюються в машинний код і завантажуються в контролер. Будь-яке завдання, яке має рішення у вигляді програми, можна реалізувати в CoDeSys.

Спочатку CoDeSys був націлений на завдання, що вимагають автономності, надійності і максимальної швидкодії при мінімізації апаратних засобів. Завдяки цьому він вийшов далеко за рамки традиційних для МЕК 61131-3 систем ПЛК. Сьогодні автомобілі, крани, екскаватори, самоскиди, яхти, друкарські машини, деревообробні верстати, ливарні і прокатні машини, складальні автомати найбільших світових брендів включають один або групу вбудованих контролерів з CoDeSys.

На сьогоднішній день CoDeSys успішно застосовується у всіх без винятку областях промисловості. У світі понад 350 компаній, виготовляють контролери з CoDeSys в якості штатного інструменту програмування. Всі конкуруючі системи відстають в рази, що дозволяє доказово говорити про світове лідерство [16].

Як продукт, CoDeSys орієнтований на виробників контролерів. Розробляючи новий контролер, вони встановлюють в нього систему виконання CoDeSys Control. З її компонентів збирають необхідну конфігурацію, додають власні ноу-хау і специфічні компоненти і отримують власне інструментальне програмне забезпечення. Як правило, до користувача CoDeSys потрапляє в

коробці разом з обладнанням. Йому потрібно тільки встановити систему і перейти до вирішення своїх практичних завдань. Всі комерційні і технічні питання, пов'язані з підтримкою ядра контролера, всіх типів його апаратних модулів, бібліотек, стеків і конфігуратор мереж його турбувати не повинні. Все це повинно бути вирішено за нього розробниками ПЛК та CoDeSys спільно.

Середовище програмування – це та частина, з якої безпосередньо має справу користувач (рисунок 2.1). Вона функціонує на ПК і є основним компонентом комплексу. Вона включає редактори для дев'яти мов програмування ПЛК, в тому числі стандартні мови МЕК 61131-3. Користувач може вибрати одну з них і програмувати простими засобами або задіяти засоби CoDeSys. На виході CoDeSys безпосередньо дає швидкий машинний код. Підтримані всі поширені сімейства мікропроцесорів від 16 до 64-розрядних.

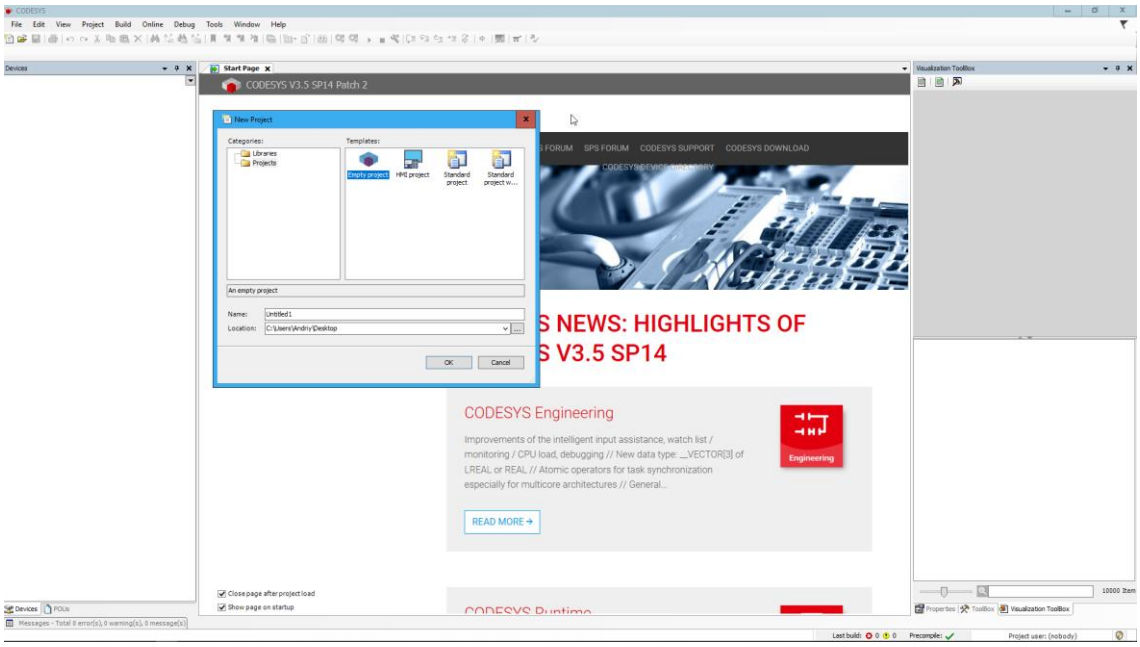

Рисунок 2.1 – Середовище CoDeSys

Середовище програмування CoDeSys включає набір інструментів для підготовки і налагодження програм, компілятори, конфігуратор, редактори візуалізації тощо. При необхідності функціональність системи доповнюється опціональними компонентами. Проект CoDeSys можна зберігати не тільки на

диску ПК, але і в контролері, якщо він має достатній обсяг пам'яті, що дозволяє уникнути втрати вихідних текстів чи плутанини в проектах. Для великих проектів передбачено використання системи контролю версій (SVN) [16].

Для налагодження користувачеві не потрібно відкривати спеціальних налагоджувальних вікон або складати будь-яких списків змінних. Безпосередньо в редакторах відображаються значення всіх видимих на екрані змінних, причому в складних виразах видно всі проміжні результати.

Комплекс CoDeSys не прив'язаний до конкретної апаратної платформи, існує кілька модифікацій, спеціально оптимізованих під різні мікропроцесори. Для прив'язки до конкретного ПЛК потрібна адаптація програми до низькорівневих ресурсів – розподілу пам'яті, інтерфейсів зв'язку та інтерфейсів уводу-виводу.

Серед особливостей середовища CoDeSys можна відзначити такі:

а) пряма генерація машинного коду, що забезпечує високу швидкодію програм користувача;

б) повноцінна реалізація мов МЕК;

в) «розумні» редактори мов побудовані в такий спосіб, що не дають робити типові для початківців помилки;

г) вбудований емулятор контролера, що дозволяє проводити налагодження проекту без додаткових апаратних засобів;

д) вбудовані елементи візуалізації дають можливість створити модель об'єкту керування та проводити налагодження проекту без виготовлення засобів імітації;

е) набір бібліотек і готових сервісних функцій.

2.2 Мови програмування в CoDeSys

Для реалізації поставлених задач у середовищі CoDeSys реалізовано такі мови програмування [17, 18]:

а) Instruction List (IL) – список інструкцій;

б) Structured Text (ST) – структурований текст.

в) Sequential Function Chart (SFC) – послідовні функціональні діаграми;

г) Function Block Diagram (FBD) – функціонально-блокові діаграми;

д) Ladder Diagram (LD) – сходинкова діаграма, мова релейно-контактних схем.

Розглянемо кожну з цих мов детальніше.

Мова IL – список інструкцій – є типовим асемблером з акумулятором та переходами за позначками. Набір інструкцій стандартизований, він не залежить від конкретної цільової платформи. Мова IL дозволяє працювати з будь-якими типами даних, викликати функції та функціональні блоки, реалізовані будь-якою мовою. Таким чином, на IL можна реалізувати алгоритм будь-якої складності, хоча текст буде досить громіздким.

Приклад коду мовою IL представлений на рисунку 2.2.

```
          LD    Speed
          GT    2000
          JMPCN VOLTS_OK
          LD    Volts
VOLTS_OK  LD    1
          ST    %Q75
```

Рисунок 2.2 – Приклад коду мовою Instruction List (IL)

Мова ST – структурований текст – це мова високого рівня. Синтаксично ST являє собою трохи адаптовану мову Паскаль. Замість процедур мови Паскаль в ST використовуються компоненти програм стандарту МЕК. Для фахівців, знайомих з мовою С, освоєння ST також не повинне викликати ніяких складностей. У більшості комплексів програмування ПЛК мова ST за замовчуванням пропонується для опису дій та умов переходів SFC. Це дійсно максимально потужне поєднання технологій, що дозволяє ефективно вирішувати будь-які задачі. Приклад коду мовою ST представлений на рисунку 2.3.

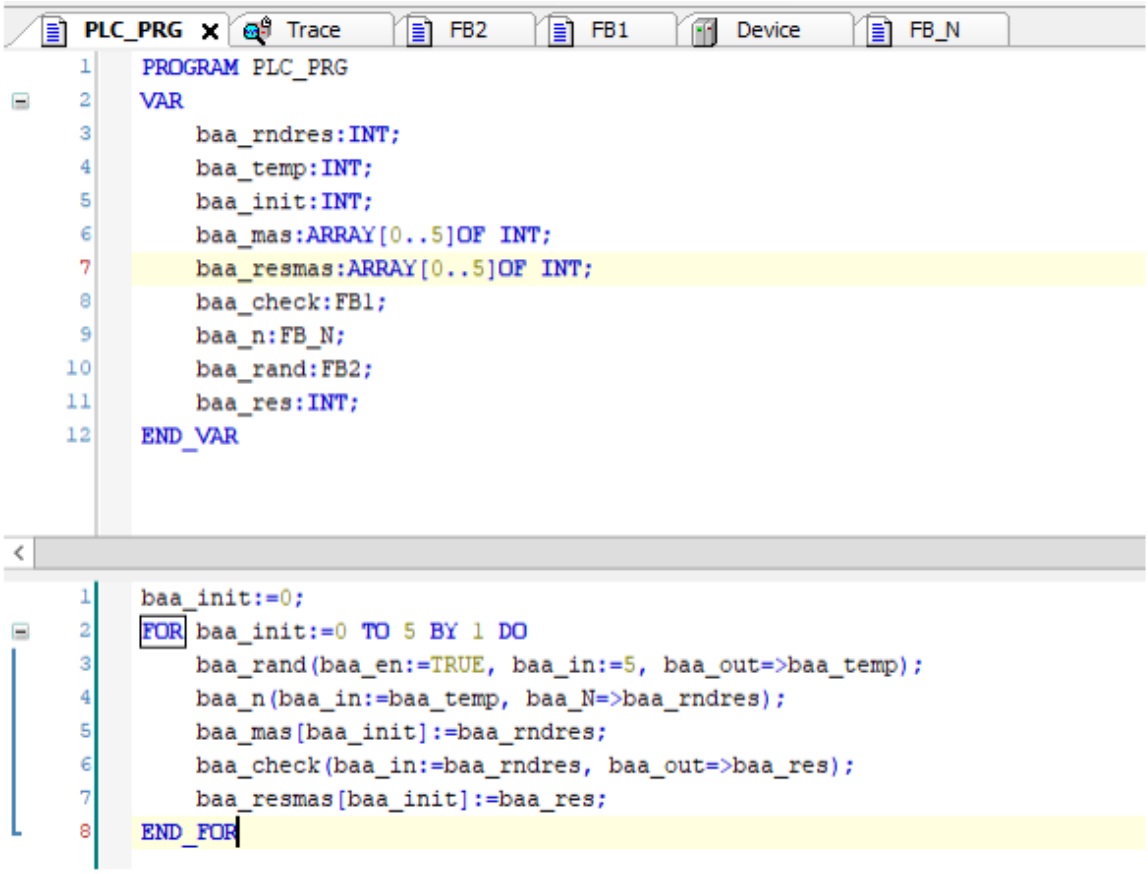

Рисунок 2.3 – Приклад коду мовою Structured Text (ST)

У сімействі мов МЕК SFC-діаграми стоять окремо, а точніше, вище відносно інших чотирьох мов. Діаграми SFC є високорівневим графічним інструментом. Графічна діаграма SFC складається із кроків і переходів між ними. Дозвіл переходу визначається умовою. Із кроком пов'язані певні дії. Опис дії виконується будь-якою мовою МЕК. Сам SFC не містить керуючих команд, дії необхідно описати на IL, ST, LD або FBD [19]. Приклад коду мовою SFC представлений на рисунку 2.4.

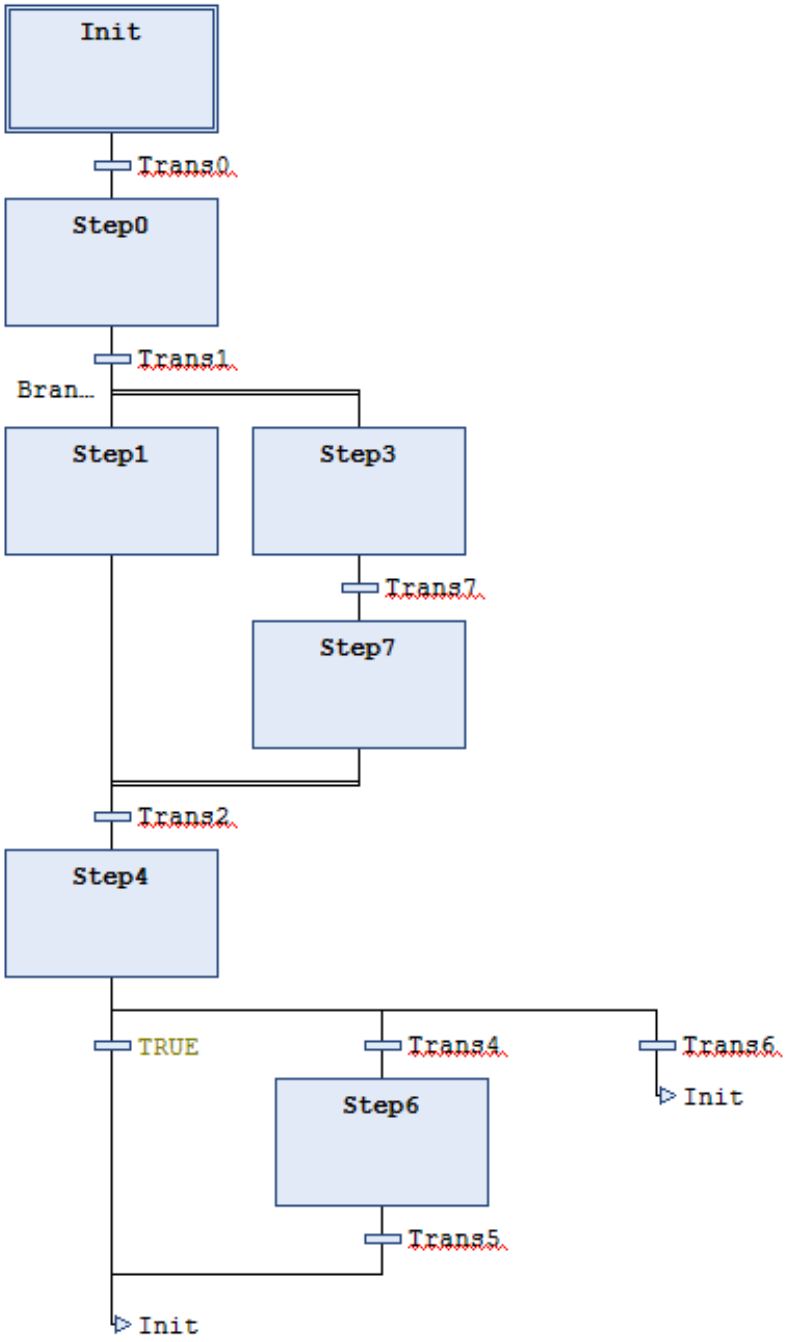

Рисунок 2.4 – Приклад Sequential Function Chart (SFC)

Мова FBD – функціонально-блокові діаграми – нагадує принципову схему електронного пристрою на мікросхемах. Провідники в FBD можуть проводити сигнали (передавати змінні) будь-якого типу (логічний, аналоговий, час тощо). Виходи блоків можуть бути підімкнені на входи інших блоків або безпосередньо на виходи ПЛК. Самі блоки, подані на схемі як «чорні ящики», можуть виконувати будь-які функції. FBD-схеми дуже чітко описують взаємозв'язок

входів і виходів діаграми. Якщо алгоритм добре описується з позиції сигналів, то його FBD-подання завжди виходить наочніше, ніж у текстових мовах. Приклад коду мовою FBD представлений на рисунку 2.5.

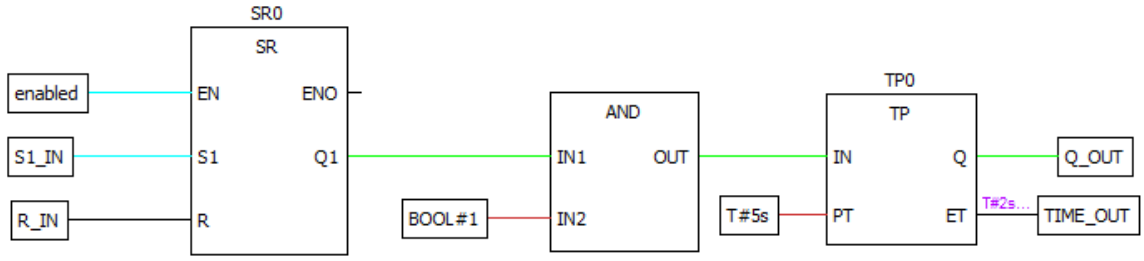

Рисунок 2.5 – Приклад Function Block Diagram (FBD)

Мова LD – мова релейно-контактних схем – графічна мова, що реалізує структури електричних ланцюгів. Графічно LD діаграма представлена у вигляді двох вертикальних шин живлення. Між ними розмішені ланцюги, утворені з'єднанням контактів. Навантаженням кожному ланцюгу служить реле. Кожне реле має контакти, які можна використати в інших ланцюгах. Послідовне (І), паралельне (АБО) з'єднання контактів та інверсія (НЕ) утворюють базис Буля. У результаті LD ідеально підходить не тільки для побудови релейних автоматів, але й для програмної реалізації комбінаційних логічних схем. Завдяки можливості включення в LD функцій і функціональних блоків, виконаних іншими мовами, сфера застосування мови практично не обмежена. Приклад коду мовою LD представлений на рисунку 2.6.

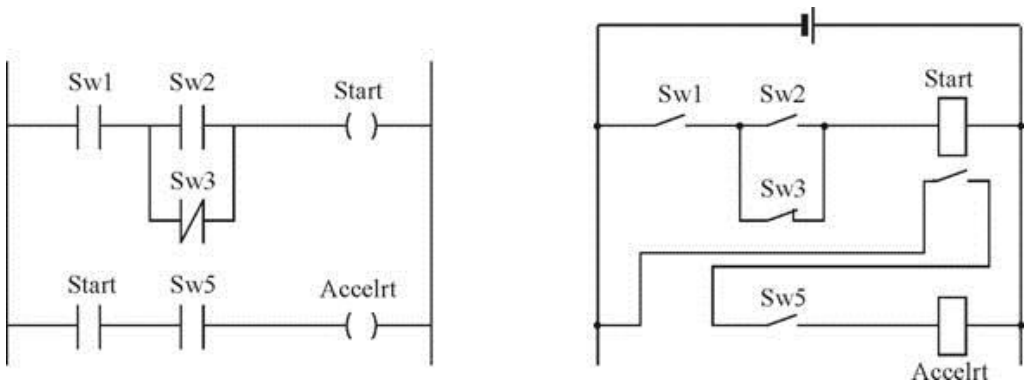

Рисунок 2.6 – Приклад Ladder Diagram (LD)

2.3 SFC діаграми для побудови дискретних автоматів

SFC-схеми керування процесами в значній мірі схожі з діаграмами станів. Діаграми станів – це графічне представлення кінцевих автоматів.

Будь-яка SFC-схема складається з елементів, що представляють кроки і умови переходів [19].

Кроки показані на схемі прямокутниками (рисунок 2.7). Реальна робота кроку (дії) описується в окремому вікні системи програмування і не відображається на діаграмі. Про призначення кроку SFC говорить тільки його назва або, якщо цього не досить, короткий текстовий опис (коментар).

Рисунок 2.7 – Графічне зображення «Кроку» мови SFC

Кроки на схемі можуть бути порожніми, що не викликає помилки при компіляції проекту. Порожні кроки є нормою при застосуванні програмування зверху вниз, характерного для SFC. Визначити дії, відповідні певному кроку, можна в будь-який час. Нема нічого дивного, якщо порожні кроки залишаться і в закінченому проекті. Завданням порожнього кроку є очікування переходу. У кожного кроку може бути три контакти – зверху і знизу для з'єднання з переходом і праворуч для з'єднання з блоком дій [20].

Нижче кроку на сполучній лінії присутня горизонтальна лінія, що позначає перехід (рисунок 2.8). Два кроки ніколи не можуть бути з'єднані безпосередньо, вони повинні завжди відділятися переходом. Умовою переходу може служити логічна змінна, логічний вираз, константа або пряма адреса.

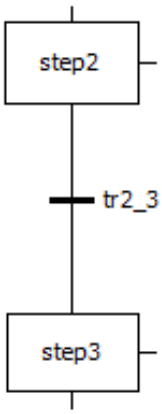

Рисунок 2.8 – Кроки, з'єднані переходом

Перехід виконується при дотриманні двох умов:

а) перехід дозволений (відповідний йому крок активний);

б) умова переходу має значення TRUE.

Прості умови відображаються безпосередньо на діаграмі праворуч від смуги, що позначає перехід (рисунок 2.9). У CoDeSys на діаграмі можна записувати тільки вираження на мові ST. Для громіздких умов застосовується інший підхід. Замість умови на діаграмі записується тільки ідентифікатор переходу. Саме ж умова описується в окремому вікні із застосуванням мови IL, ST, LD або FBD. Змінні або прямі адреси використовуються в умови переходу тільки для читання. В умовному вираженні переходу не можна викликати екземпляри функціональних блоків і використовувати операцію присвоювання.

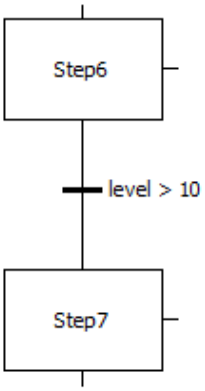

Рисунок 2.9 – Перехід між кроками з визначеною умовою

Ознакою того, що ідентифікатор переходу на діаграмі є окремо реалізованим умовою, а не простою логічною змінною, служить зафарбований кут переходу. В якості умови переходу може бути задана логічна константа. Якщо задано TRUE, то крок буде виконаний одноразово, за один робочий цикл, далі управління перейде до наступного кроку. Якщо задана умова FALSE, то крок буде виконуватися нескінченно.

Кожна SFC-схема починається з кроку, виділеного графічно подвійними вертикальними лініями або по всьому периметру (рисунок 2.10). Це – початковий крок. Найменування початкового кроку може бути довільним (за замовчуванням Init). Початковий крок присутній обов'язково, хоча і може бути порожнім.

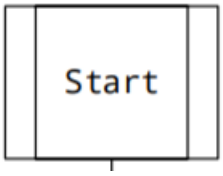

Рисунок 2.10 – Початковий крок

Кілька гілок SFC можуть бути паралельними (рисунок 2.11). Ознакою паралельних гілок на схемі є подвійна горизонтальна лінія. Кожна паралельна гілка починається і закінчується кроком. Тобто умова входу в паралельність завжди одне, умова виходу теж одне на всіх.

Паралельні гілки виконуються теоретично одночасно. В житті це означає – в одному робочому циклі, зліва направо. Умова переходу, що завершує паралельність, перевіряється тільки в разі, якщо в кожної паралельної гілки активні останні кроки.

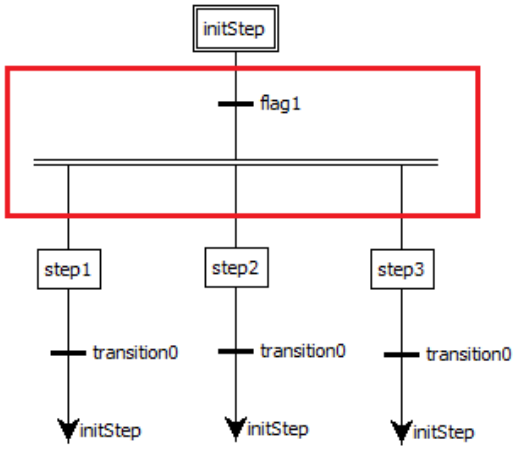

Рисунок 2.11 – Фрагмент схеми з трьома паралельними гілками

Кілька гілок SFC можуть бути альтернативними гілками. Ознакою альтернативних гілок на схемі є одинарна горизонтальна лінія (рисунок 2.12). Кожна альтернативна гілка починається і закінчується власною умовою переходу. Перевірка альтернативних умов виконується зліва направо. Якщо вірна умова знайдена, то інші альтернативи не розглядаються. В альтернативних гілках завжди працює тільки одна з гілок, тому її закінчення і буде означати перехід до наступного за альтернативною групою кроцу. При створенні альтернативних гілок бажано ставити взаємовиключні умови. У цьому випадку ймовірність допустити помилку при аналізі або в процесі доопрацювання діаграми значно нижче.

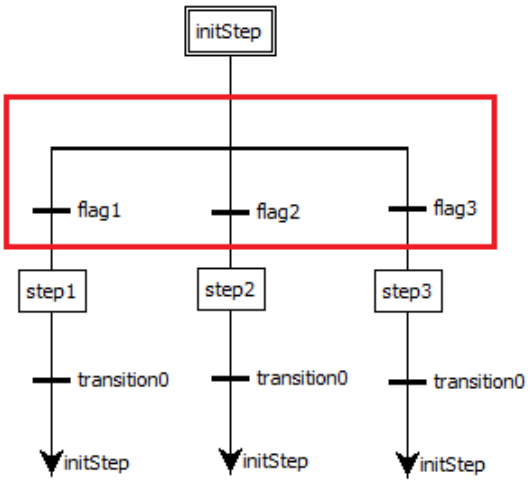

Рисунок 2.12 – Фрагмент схеми з альтернативними гілками

ВИСНОВКИ ДО РОЗДІЛУ 2

У другому розділі дипломного проекту було розглянуто середовище CoDeSys (Controller Development System) – інтегроване середовище розробки (IDE) додатків для програмованих контролерів. На сьогоднішній день CoDeSys – це найпопулярніший в світі апаратно незалежний комплекс для прикладного програмування ПЛК та вбудованих контролерів. Основним його компонентом є середовище програмування на мовах стандарту МЕК 61131-3.

В середовищі CoDeSys реалізовано такі мови програмування, як IL, ST, SFC, FBD та LD. Вони дозволяють просто, ефективно та максимально швидко реалізовувати поставлені задачі. Вбудовані елементи візуалізації дають можливість створити модель об'єкту керування та проводити налагодження проекту без виготовлення засобів імітації.

Були розглянуті SFC діаграми, які є незамінними для побудови дискретних автоматів. У розділі наведені приклади основних елементів SFC-схем, що використовуються для моделювання.

3 ПРИКЛАД МОДЕЛЮВАННЯ ДИСКРЕТНОГО АВТОМАТУ В CODESYS

3.1 Опис об’єкту

Прикладом для моделювання дискретного автомату було обрано систему керування генераторною установкою (рисунок 3.1).

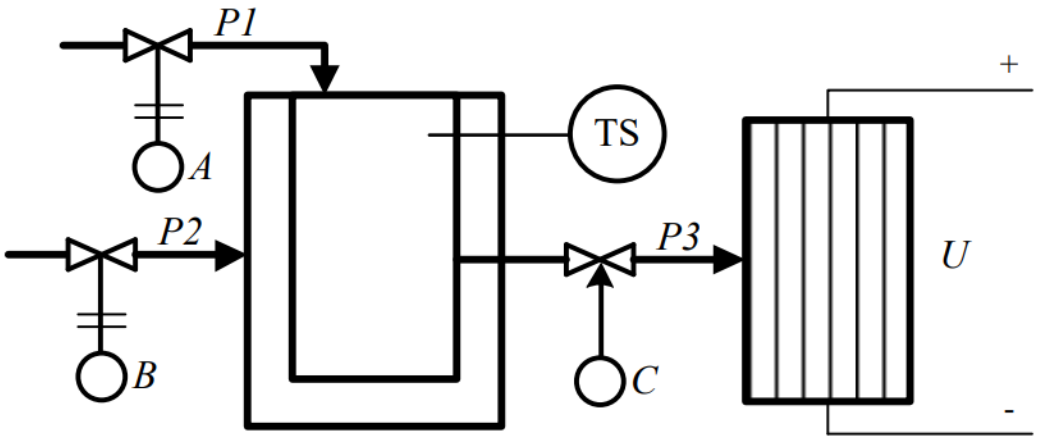

Рисунок 3.1 – Генераторна установка

Опис установки: клапан А повертається на 95% і починається процес наповнення ємності речовиною. Наповнення триває 60 хвилин. Потім починається процес нагрівання відкриттям клапану В. Нагрівання відбувається протягом 40 хвилин. Після того, як температура досягла потрібного значення, клапан С відкривається і починається розгін турбіни U паром Р3.

3.2 Структурна схема дискретного автомату

Діаграми станів дозволяють чітко визначити набір функцій конкретного пристрою, спростити процес проектування і розробки програмного забезпечення, а також визначити загальну ефективність обладнання. Діаграми станів є графічним представленням кінцевих автоматів. Зручністю використання кінцевих автоматів є простота їх опису. Для використання автомата необхідна наступна інформація:

а) кінцева множина станів S. Кожен стан є унікальним і значущим в рамках даної системи. В SFC стан представляється у вигляді кроку;

б) множина подій E. Події обумовлюють переходи системи з одного стану в інший. В рамках SFC події можна асоціювати з вхідними сигналами системи (сигнали польового рівня, команди оператора і т.д.), а також внутрішніми змінними, залежними від закладених алгоритмів (наприклад, результат роботи таймера);

в) початковий стан S_0. Один із станів множини S має бути початковим. В рамках SFC початковий стан (початковий крок) зазвичай відповідає ініціалізації програми управління;

г) множина фінальних станів F. Ці стани належать множині S і відповідають завершенню технологічного процесу. Якщо автомат після припинення роботи знаходиться в стані, що не належить F, то це свідчить про некоректну роботу системи управління;

д) множина умов переходів T. Умова переходу є функцією, що зв'язує три елементи: $<s_i, e_{ij}, e_j>$, де S_i – поточний стан автомата, e_{ij} – подія, що визначає перехід від стану S_i до стану S_j, S_j – новий стан автомата.

Модель виконання кінцевого автомата має на увазі, що в кожен момент часу автомат знаходиться в одному з станів. При цьому його активний стан періодично змінюються. У початковий момент часу автомат знаходиться в стані S_0. Він залишається в цьому стані до виникнення події, яка активує перехід до

наступного стану. Далі в міру виникнення тих чи інших подій автомат буде переходити в відповідні стани, поки його поточний стан не стане фінальним. Таким чином, кінцевий автомат можна представити у вигляді алгоритму, вхідними даними якого є події, а вихідними – інформація про поточний активний стан або некоректне завершення роботи.

На основі вибраної системи керування мною було розроблено діаграму станів, зображену на рисунку 3.2.

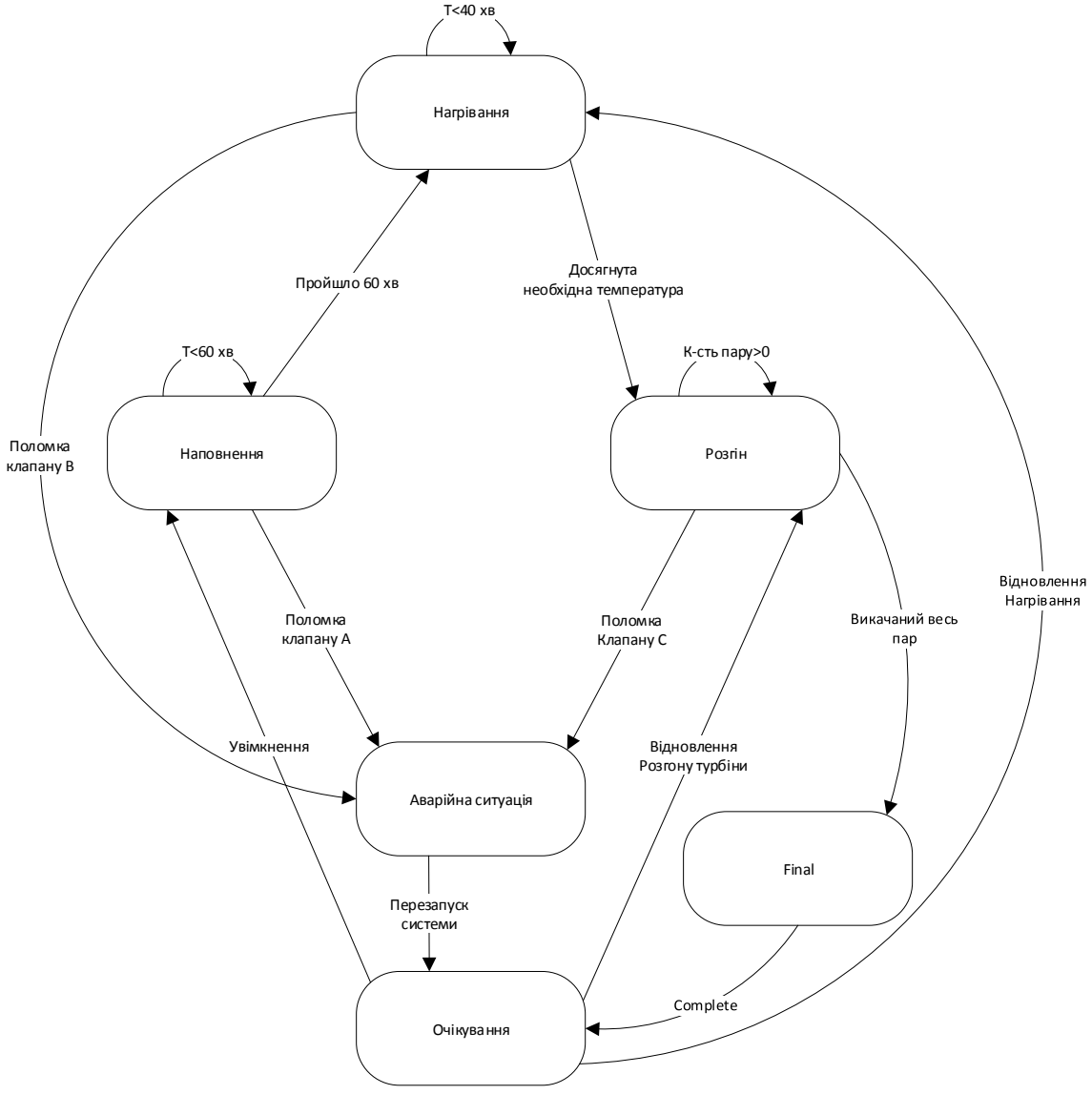

Рисунок 3.2 – Діаграма станів

3.3 Дискретний автомат на SFC

Діаграма станів являється лише описом процесу управління. Для створення алгоритму управління необхідно використовувати відповідне програмне обладнання, в якому буде здійснено реалізацію цієї діаграми. Одним з варіантів реалізації є використання мови SFC. Завдяки своїй простоті і наочності його не складе труднощів освоїти інженеру будь-якої спеціальності незалежно від рівня його кваліфікації. Саме цей варіант реалізований у даному дипломному проекті (рисунок 3.3).

Для програмування дискретного автомату, діаграма станів якого зображена на рисунку 3.2, було реалізовано такі блоки SFC-діаграми:

а) Init – блок ініціалізації, з якого починається будь-яка SFC-діаграма;

б) Waiting – блок очікування ввімкнення генератора;

в) Aborting – блок вимкнення установки;

г) Reset_system – блок перезапуску установки;

д) Open_A – блок відкриття клапану А (початок процесу наповнення ємності рідиною);

е) Open_B – блок відкриття клапану В (початок процесу нагрівання рідини), закриття клапану А (завершення процесу наповнення);

ж) Close_B – блок закриття клапану В (завершення процесу нагрівання);

з) Open_C – блок відкриття клапану С (початок процесу розгону турбіни парою);

и) Final_state – блок закриття клапану С (успішне завершення роботи установки).

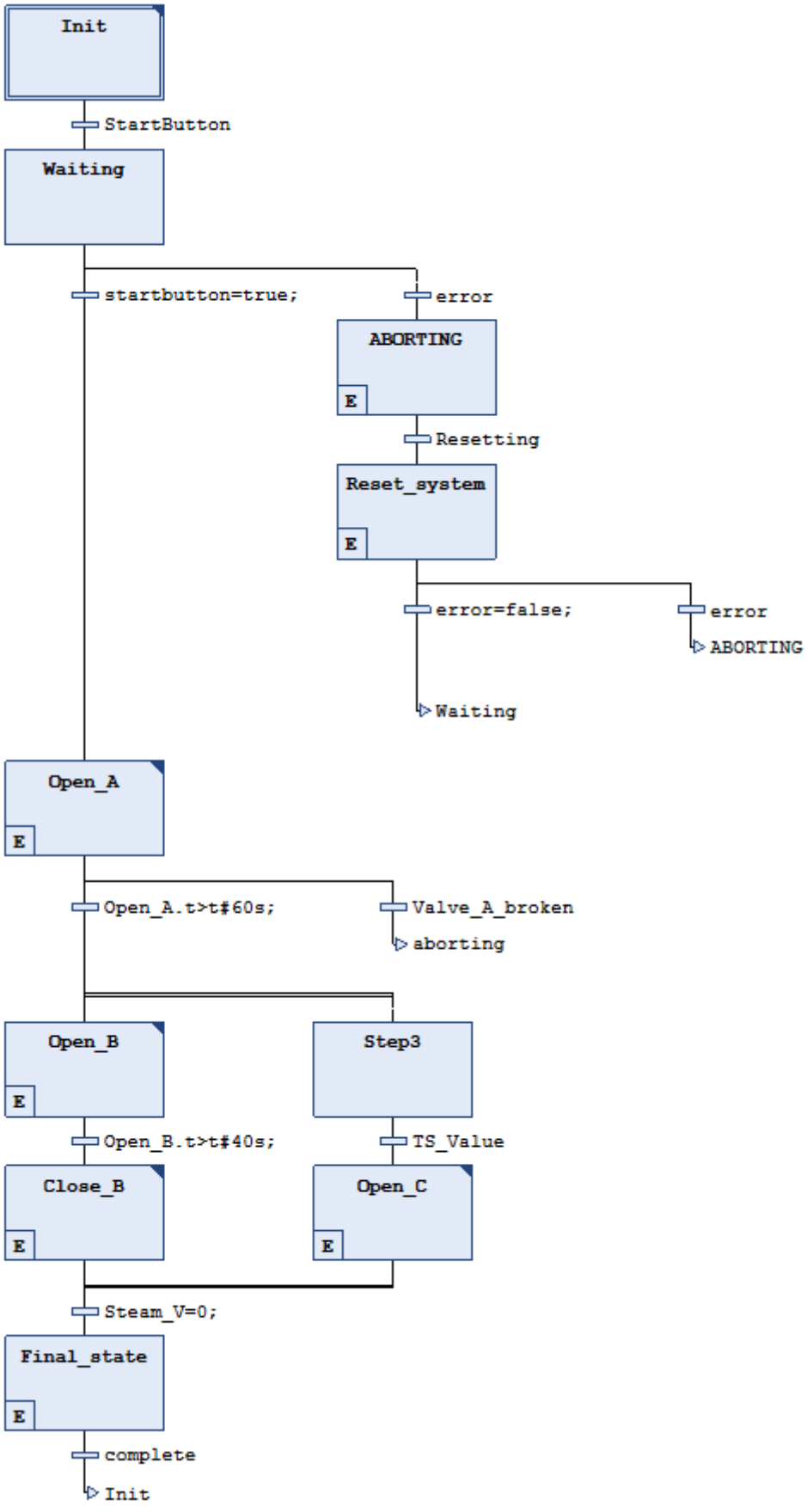

Рисунок 3.3 – SFC-діаграма

Далі будуть описані всі кроки програми у відповідності до послідовності кроків представленій раніше.

Виконання програми починається з блоку ініціалізації змінних програми Init, потім програма переходить в стан очікування і продовжується лише коли буде поданий сигнал про початок роботи.

Після отримання дозволу на початок роботи програма дає сигнал на відкриття клапану A на 95%, для початкового наповнення баку мішалки, це відбувається в блоці Open_A.

Програма перейде на наступний крок виконання через 60 хвилин і дасть команду на початок нагрівання через відкриття клапану B (блок Open_B). Нагрівання триватиме 40 хвилин. Перехід на наступний крок відбудеться тільки в тому випадку, коли датчик TS покаже, що необхідна температура досягнута. Тоді відкриється клапан C (блок Open_C) і почнеться розгін турбіни паром. Коли весь пар викачаний – система переходить в кінцевий стан (блок Final_state) і клапан C закриється.

3.4 Імітаційне моделювання системи керування в CoDeSys

Розглянемо етап виконання програми, коли з клапаном A відбулась поломка (рисунок 3.4). Як видно з рисунку, клапан A вийшов із ладу. Про це свідчить значення TRUE змінної Valve_A_broken. Поки клапан A знаходиться в аварійному стані, система не зможе продовжити функціонування і буде переходити між блоками ABORTING і Reset_system, поки не буде активований перемикач Fix button.

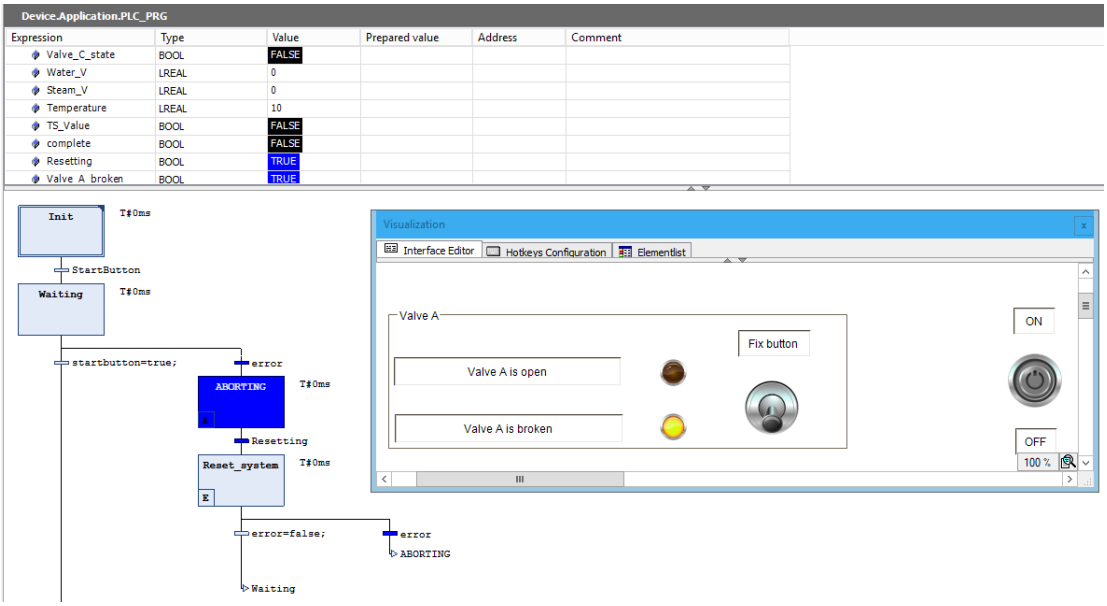

Рисунок 3.4 – Аварійна ситуація клапану А

Щоб продовжити роботу генераторної установки, нам потрібно виправити поломку. На рисунку 3.5 видно, що перемикач Fix button активували, клапан А відремонтований (змінна Valve_A_broken змінила своє значення на False) і програма перейшла в блок очікування.

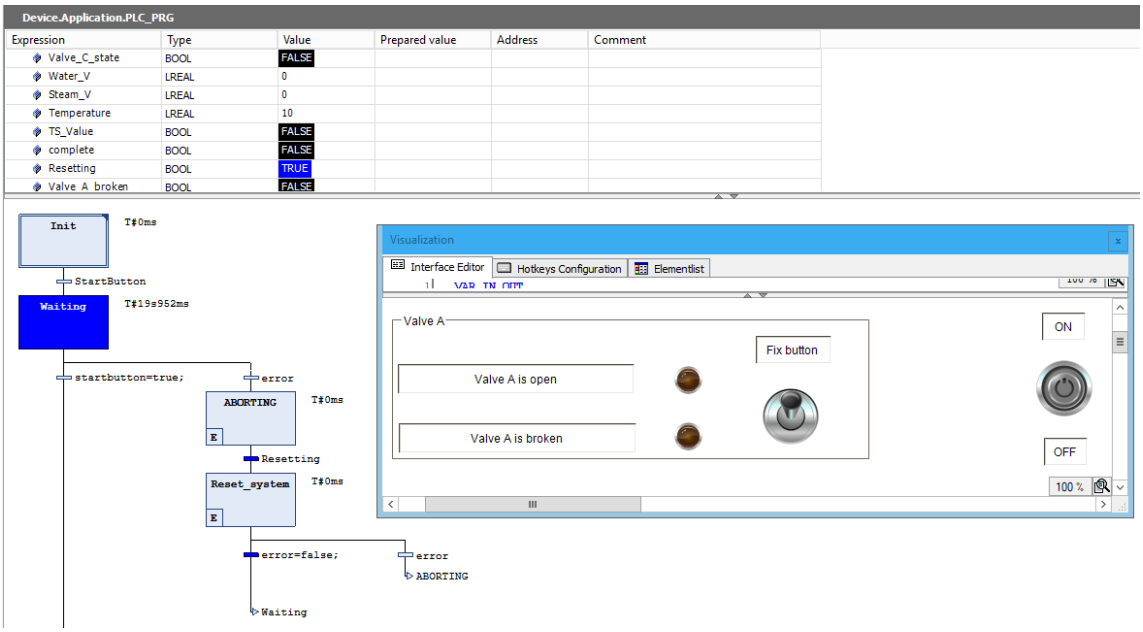

Рисунок 3.5 – Очікування запуску установки після усунення неполадки

На рисунку 3.6 зображено подальше функціонування системи. Аварійна ситуація ліквідована і через клапан А відбувається наповнення ємності рідиною. З рисунку видно, що протягом 33,6 секунд після відкриття клапану А ємність наповнилася рідиною на 33,62 м^3.

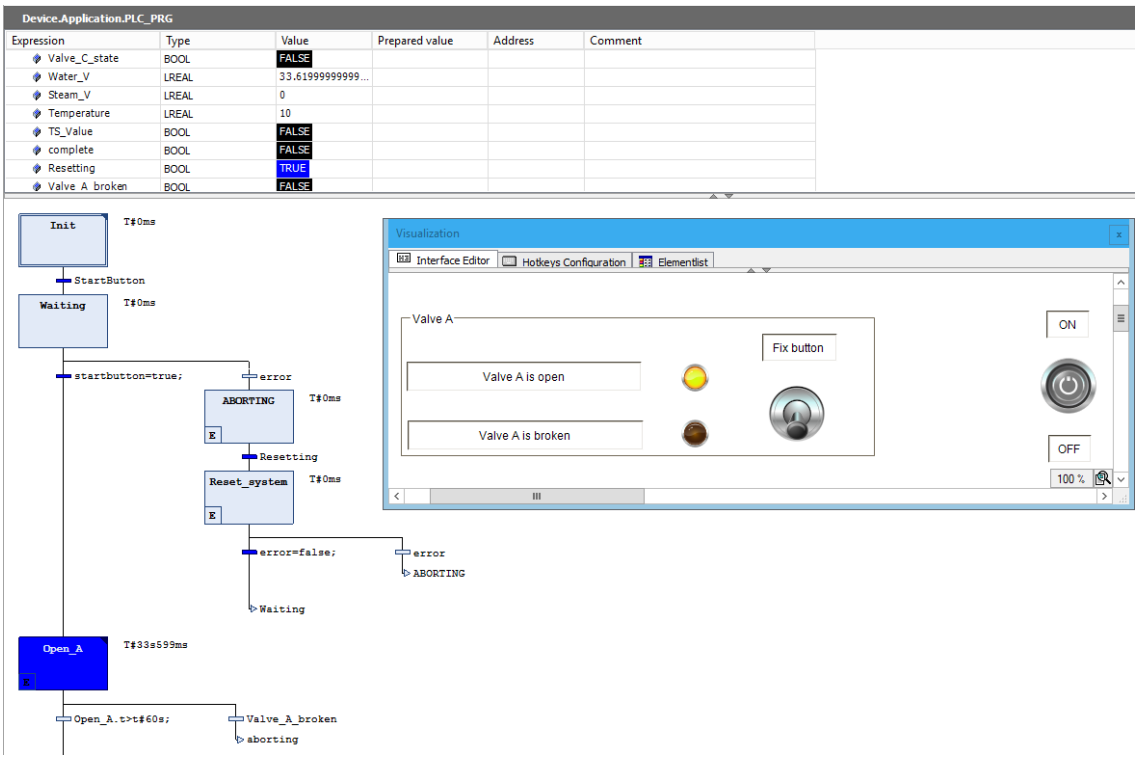

Рисунок 3.6 – Скріншот роботи програми (наповнення)

На рисунку 3.7 продемонстровано процес нагрівання рідини, поки датчик TS ще не спрацював. З рисунку видно, що за 18,7 секунд роботи клапану В температура зросла з 10 °С до 89 °С і при цьому утворилося 3,6 м^3 пару. Об’єм рідини, що при цьому залишилася – 49,1 м^3.

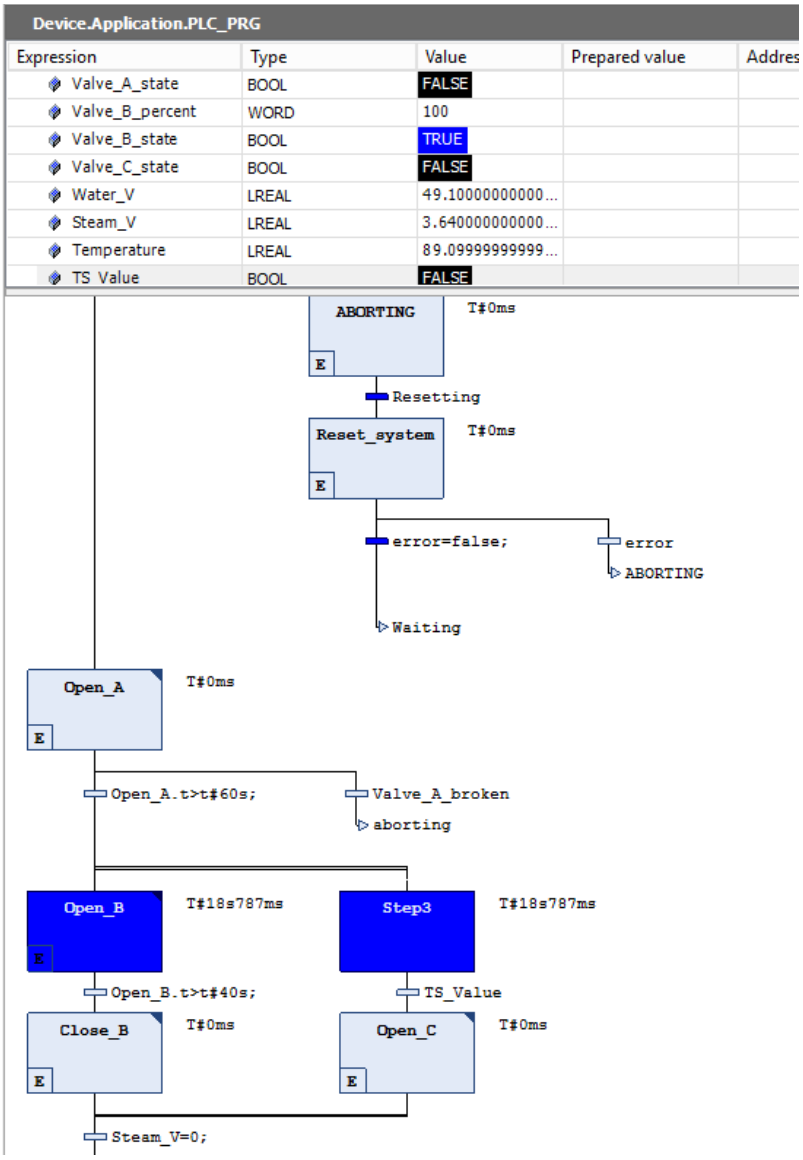

Рисунок 3.7 – Скріншот роботи програми (нагрівання)

Розглянемо наступний етап функціонування генераторної установки, зображений на рисунку 3.8. З рисунку видно, що датчик TS спрацював, клапан C відкрився і розпочався розгін турбіни парою. Процес нагрівання все ще триває. Температура досягла 140 °C, рідини в резервуарі залишилося 26,9 м^3, а об’єм пару в ємності в даний момент часу становить 20,5 м^3.

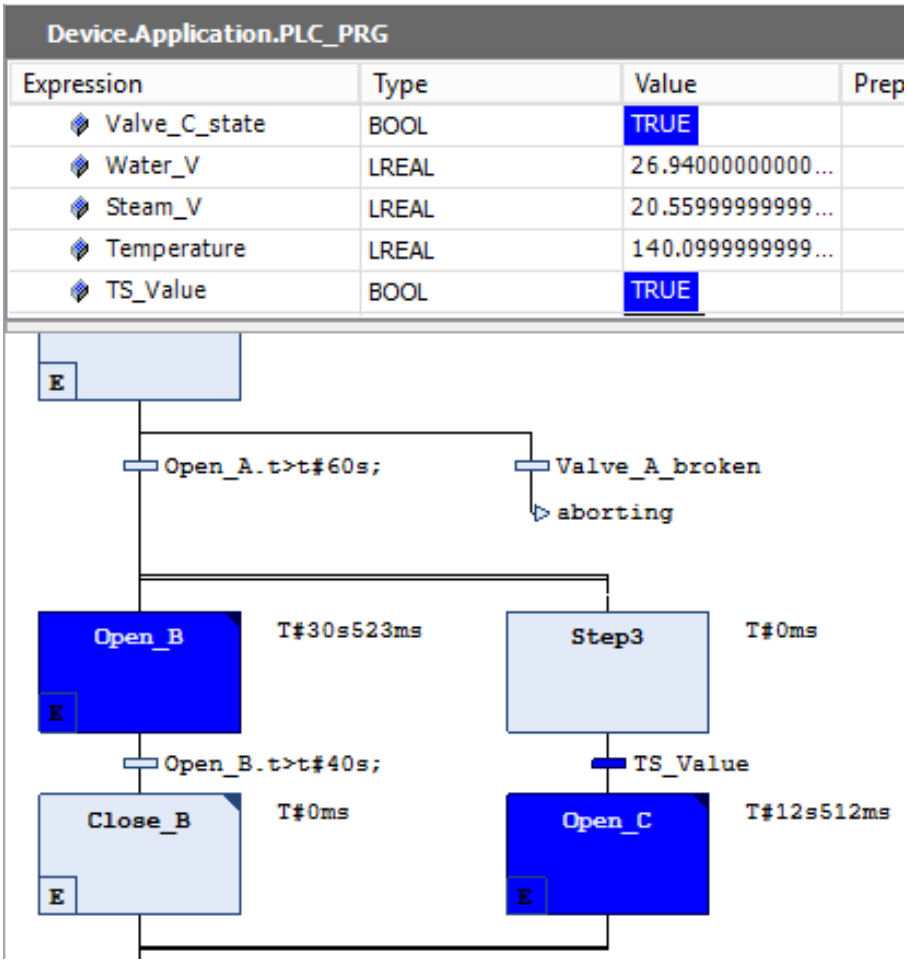

Рисунок 3.8 – Скріншот роботи програми (розгін турбіни)

Запустимо нашу систему ще раз і подивимось на виконання поставленого завдання, відслідковуючи графіки зміни об’єму рідини, температури рідини і кількості пару. Спочатку розглянемо як змінюється об’єм рідини в ємності (рисунок 3.9).

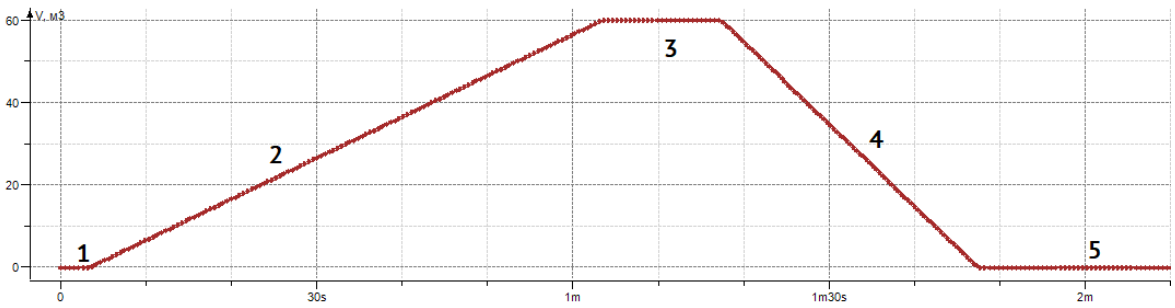

Рисунок 3.9 – Графік зміни об’єму рідини в резервуарі

На даному графіку можна виділити 5 етапів.

Перший етап – очікування відкриття клапану А. В резервуарі немає рідини.

Другий етап – клапан А відкритий і в резервуар поступає рідина протягом 60 хвилин (для наочності взято 60 секунд).

Третій етап – починається процес нагрівання. Кількість рідини не змінюється.

Четвертий етап – початок випаровування. Кількість рідини починає зменшуватись.

П’ятий етап – температура в резервуарі опустилась до рівня, коли рідина перестає випаровуватись.

Перейдемо до розгляду зміни температури в ємності (рисунок 3.10).

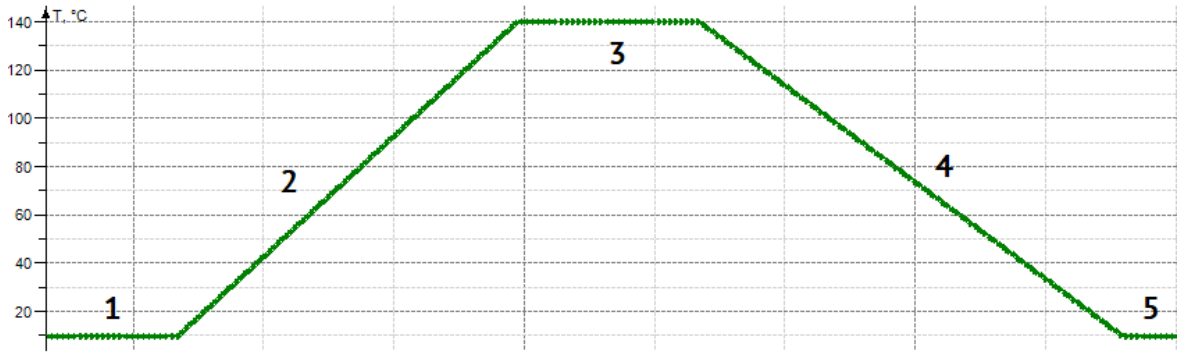

Рисунок 3.10 – Графік зміни температури

На даному графіку можна також виділити 5 етапів.

Перший – процес нагрівання ще не запущено і температура рідини не змінюється.

Другий – відкрився клапан В і розпочалось нагрівання.

Третій – рідина досягла температури 140 °С, клапан В все ще відкритий.

Четвертий – минуло 40 хвилин, відведених на нагрівання. Клапан В закритий і рідина починає охолоджуватись.

П’ятий – рідина охолола до температури 20 градусів.

Розглянемо графік зміни кількості пару в резервуарі (рисунок 3.11).

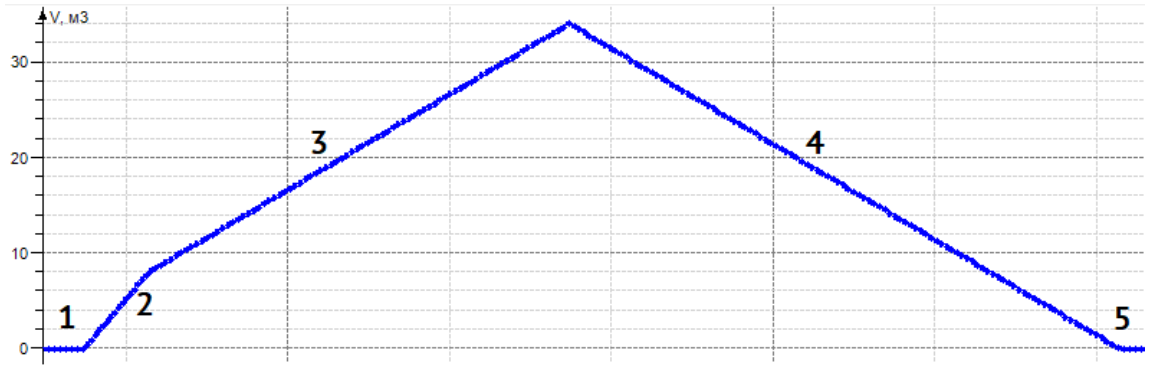

Рисунок 3.11 – Графік зміни кількості пару

Як і на двох попередніх графіках, можна виділити 5 етапів.

Перший – кількість пари рівна нулю. Тобто рідина ще не достатньо нагрілась.

Другий – температура досягла певного значення. Починає випаровуватись рідина.

Третій – спрацював датчик TS. Відкрився клапан C і почався розгін турбіни. Нагрівання все ще продовжується.

Четвертий – рідина перестала випаровуватись. Розгін турбіни триває.

П’ятий – весь пар викачано із резервуару. Система закінчила свою роботу.

На рисунку 3.12 представлено об’єднані графіки. Графік синього кольору Steam_V фіксує залежність об’єму пари в ємності від часу. Графік зеленого кольору Temperature фіксує залежність температури в резервуарі від часу. Графік червоного кольору Water_V фіксує залежність об’єму рідини в ємності від часу.

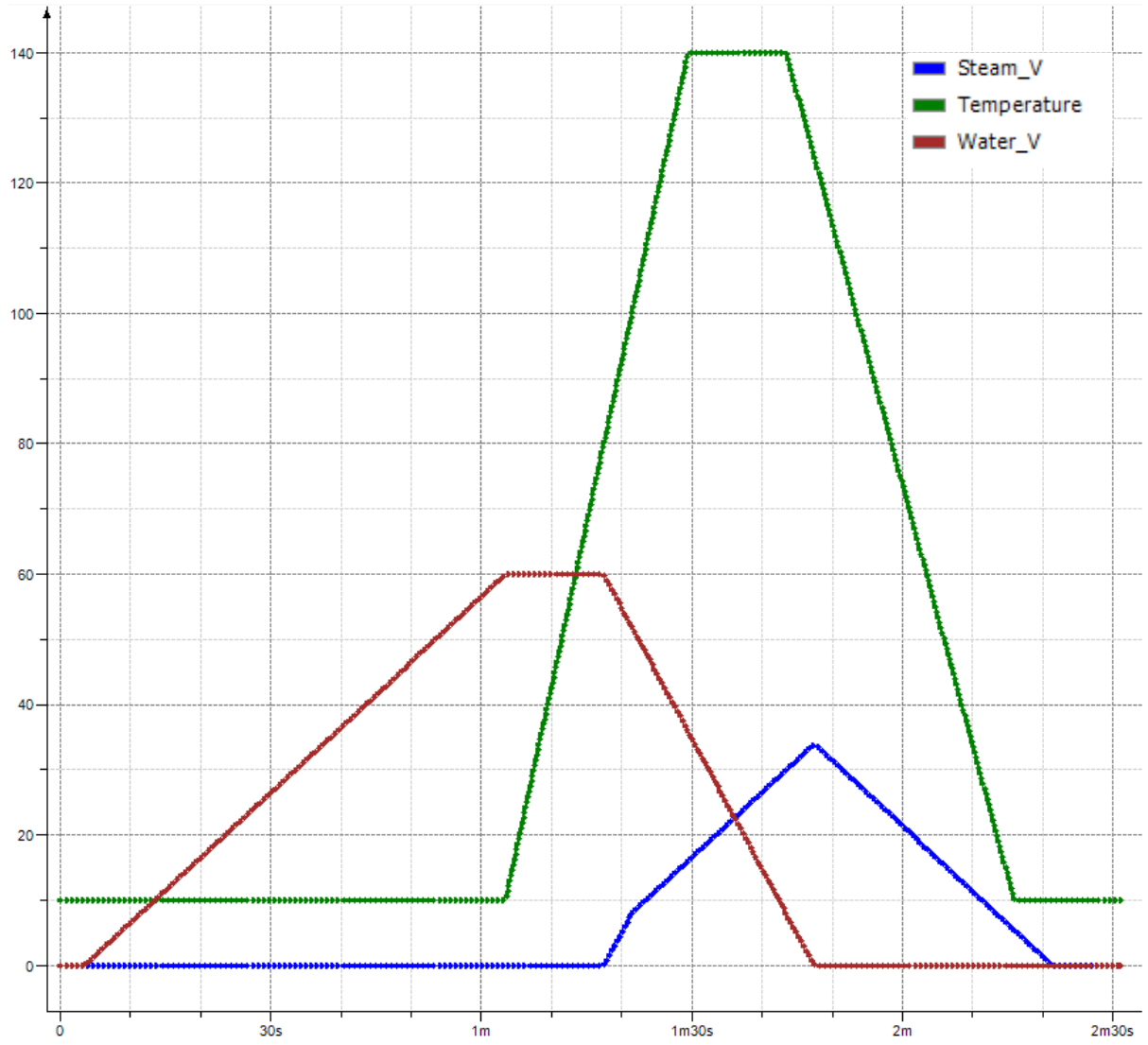

Рисунок 3.12 – Загальний графік роботи системи

З наведеного вище графіку видно, що вся рідина, якою наповнили резервуар через клапан А, випарувалась. Весь пар, що утворився при цьому, був використаний для розгону турбіни генераторної установки. При цьому температура в ємності досягла максимального значення в 140 °С.

ВИСНОВКИ ДО РОЗДІЛУ 3

Під час роботи над даним розділом дипломного проекту була створена модель дискретного автомату. Прикладом об’єкту було обрано систему керування генераторною установкою. Була сформульована задача та були описані всі етапи моделювання автомату.

Одним з основних кроків була побудова структурної схеми дискретного автомату. Мною була створена діаграма станів, на якій відображено кожен етап роботи установки.

Наступним кроком було створення SFC-схеми дискретного автомату. Вона була побудована за допомогою інструментів роботи з мовою SFC середовища розробки CoDeSys.

Останнім етапом було імітаційне моделювання системи керування в CoDeSys з наведенням графіків процесів. Особливості роботи з даною IDE були розглянуті у попередньому розділі.

Результатом роботи є працююча модель системи керування генераторною установкою в CoDeSys.

ВИСНОВКИ

У процесі написання дипломного проекту було встановлено, що існуючі сьогодні вимоги до програмного забезпечення та швидкі темпи розвитку технічних систем спричиняють зростання інтересу до теорії автоматів та до дискретних автоматів у цілому. Завдяки цьому з'являється можливість вирішувати все складніші задачі і, одночасно, досягати все більшої ефективності.

У першому розділі роботи розглядається теорія дискретних автоматів, що вивчає математичні моделі перетворювачів дискретної інформації – автоматів. Вона вирішує такі основні задачі, як аналіз і синтез автоматів, визначення повноти, мінімізація та еквівалентні перетворення автоматів. Були розглянуті основні моделі, що описують функціонування абстрактних автоматів. Серед них – автомати Мілі, автомат Мура, С-автомат та мережі Петрі. Також були вивчені та розглянуті на конкретному прикладі методи задання дискретних автоматів – таблиці переходів, графи та матриці.

Другий розділ дипломного проекту вичерпно описує інтегроване середовище розробки додатків для програмованих контролерів CoDeSys (Controller Development System) – найпопулярніший в світі апаратно незалежний комплекс для прикладного програмування ПЛК та вбудованих контролерів. основним компонентом якого є середовище програмування на мовах стандарту МЕК 61131-3. Був приведений повний перелік мов та засобів, які використовуються в CoDeSys для реалізації задач. Також були розглянуті основні принципи створення важливого компоненту процесу моделювання дискретного автомату – SFC-діаграми. Вони стоять вище відносно інших чотирьох мов та є високорівневим графічним інструментом.

Спираючись на результати, отримані у попередніх розділах, в останньому розділі дипломного проекту була створена модель дискретного автомату в програмному комплексі CoDeSys. Прикладом для моделювання дискретного автомату було обрано систему керування генераторною установкою. Для цього

була створена діаграма станів, яка дозволила визначити набір необхідних функцій та визначити загальну ефективність обладнання. Після цього для розробки алгоритму управління була створена діаграма з використанням мови SFC, на основі необхідних даних, отриманих при попередніх кроках.

Останнім кроком було імітаційне моделювання системи керування генераторною установкою. Були детально розглянуті всі кроки роботи установки, та на їх основі побудовані графіки, що відображають стан системи. Загальний результат зображений на суміщеному графіку, по якому можна легко прослідкувати стан автомату в будь-який момент часу. Отримана модель і є основним результатом даного дипломного проекту.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Гуренко В.В. Введение в теорию автоматов / В.В. Гуренко. – МГТУ им. Н. Э. Баумана, 2013. – 62 с.
2. Лупал А.М. Теория автоматов : учеб. пособие / А.М. Лупал – СПбГУАП. СПб., 2000 – 119 с.
3. Кудрявцев В.Б. Теория автоматов : учебник для бакалавриата и магистратуры / В.Б. Кудрявцев, С.В. Алешин, А.С. Подколзин. — 2-е изд., испр. и доп. — Москва : Издательство Юрайт, 2018. — 320 с.
4. Глушков В.М. Синтез цифровых автоматов / В.М. Глушков — М.: Государственное издательство физико-математической литературы, 1962. — 476 с.
5. Хопкрофт Дж. Введение в теорию автоматов, языков и вычислений / Introduction to Automata Theory, Languages, and Computation. — М.: Вильямс, 2002. — 528 с.
6. Касьянов В.Н. Лекции по теории формальных языков, автоматов и сложности вычислений / В.Н. Касьянов— Новосибирск: НГУ, 1995. — 112 с.
7. Белоусов А.И. Дискретная математика / А.И. Белоусов, С. Б. Ткачев — М.: МГТУ, 2006. — 744 с.
8. Хопкрофт Дж. Дискретная математика / Дж. Хопкрофт, Р. Мотвани, Дж. Ульман. — 2-е изд. — Вильямс, 2002. — 528 с.
9. Серебряков В.А. Теория и реализация языков программирования / В.А. Серебряков, М.П. Галочкин, Д.Р. Гончар, М.Г.Фуругян — М.: МЗ-Пресс, 2006 г., 2-е изд. – 358 с.
10. Романов В.Ф. Лекции по теории автоматов / В. Ф. Романов – 1 ч. - Владимир, 2009. – 26 с.
11. Кочубей О.О. Прикладна теорія цифрових автоматів. Логічні основи / О.О. Кочубей, О.В. Сопільник – Дніпропетровськ : Дніпропетровськ : Вид-во ДНУ : РВВ ДНУ, 2009. – 264 с.

12. Харари Ф. Теория графов / Ф. Харари – М.: Мир, 1973. – 300 с.

13. Фон Нейман Дж. Теория самовоспроизводящихся автоматов / Дж. Фон Нейман – Пер. с англ. – М.: Мир, 1971. – 384 с.

14. Карацуба А. А. Решение одной задачи из теории конечных автоматов / А. А. Карацуба – УМН, т. 15, № 3(93), 1960. – с. 157—159.

15. Карпов Ю.Г. Теория автоматов: Учеб. для вузов / Ю.Г. Карпов – М.: Питер, 2003, 2002. – 208с.

16. CODESYS V3 : Документация [Електронний ресурс] / Доступ : https://help.codesys.com/webapp/_cds_f_development_system_introduction;product=codesys;version=3.5.10.0

17. ГОСТ Р МЭК 61131-3-2016 Контроллеры программируемые.

18. Золотарев С.В. CoDeSys – интегрированный комплекс МЭК 61131-3 программирования / С.В. Золотарев, И.В. Петров – НАУЧТЕХИЗДАТ 2005г. – 8 с.

19. Петров И. В. Программируемые контроллеры. Стандартные языки и приёмы прикладного проектирования / И.В. Петров — М.: СОЛОН-Пресс, 2004. — 256 с.

20. Общие сведения о языке SFC [Електронний ресурс] / Copyright 2018, JSC "INEUM named after I.S.Bruk". Доступ : https://sm1820.github.io/beremiz/iec_guide/sfc_guide.html